UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
FACULDADE DE CIÊNCIAS - CAMPUS BAURU
DEPARTAMENTO DE COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

GUILHERME CAMARGO DE OLIVEIRA

OTIMIZAÇÃO EVOLUCIONISTA DE HIPERPARÂMETROS PARA MÁQUINAS DE BOLTZMANN RESTRITAS CONVOLUCIONAIS

BAURU
Junho/2020

GUILHERME CAMARGO DE OLIVEIRA

OTIMIZAÇÃO EVOLUCIONISTA DE HIPERPARÂMETROS PARA MÁQUINAS DE BOLTZMANN RESTRITAS CONVOLUCIONAIS

Trabalho de Conclusão de Curso do Curso de Ciência da Computação da Universidade Estadual Paulista "Júlio de Mesquita Filho", Faculdade de Ciências, Campus Bauru.
Orientador: Prof. Dr. João Paulo Papa
Coorientador: Gustavo Henrique de Rosa

BAURU
Junho/2020

Guilherme Camargo de Oliveira Otimização Evolucionista de Hiperparâmetros para Máquinas de Boltzmann Restritas Convolucionais/ Guilherme Camargo de Oliveira. – Bauru, Junho/2020- 70 p. : il. (algumas color.) ; 30 cm.
Orientador: Prof. Dr. João Paulo Papa
Trabalho de Conclusão de Curso – Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Ciência da Computação, Junho/2020.
1. Metaheurística 2. Máquina de Boltzmann Restrita Convolucional 3. Programação Genética 4. Algoritmo Genético 5. Otimização de hiperparâmetros 6. Rede Neural Artificial 7. Inteligência Artificial

Guilherme Camargo de Oliveira

Otimização Evolucionista de Hiperparâmetros para Máquinas de Boltzmann Restritas Convolucionais

Trabalho de Conclusão de Curso do Curso de Ciência da Computação da Universidade Estadual Paulista "Júlio de Mesquita Filho", Faculdade de Ciências, Campus Bauru.

Banca Examinadora

Prof. Dr. João Paulo Papa
Orientador
Departamento de Ciência da Computação
Faculdade de Ciências
Universidade Estadual Paulista "Júlio de Mesquita Filho"

Profª. Drª. Simone das Graças Domingues Prado
Departamento de Ciência da Computação
Faculdade de Ciências
Universidade Estadual Paulista "Júlio de Mesquita Filho"

Prof. Dr. Kelton Augusto Pontara da Costa
Departamento de Ciência da Computação
Faculdade de Ciências
Universidade Estadual Paulista "Júlio de Mesquita Filho"

Bauru, _____ de _____ de _____.

Agradecimentos

Agradeço a minha família por todo o carinho e apoio durante esses anos, especialmente aos meus pais e ao meu irmão. Quero dedicar essa conquista a eles, que estiveram desde o início, acompanhando meu crescimento e incentivando meus estudos. O meu pai, Sergio Oliveira, por sempre me aconselhar, a minha mãe, Marcia Camargo, me incentivando a sempre enfrentar os problemas e o meu irmão, Gabriel Oliveira, por suporte e companheirismo. Meus pais trabalharam muito para que eu chegasse até aqui, deram a oportunidade de me dedicar aos estudos e assim, passar o conhecimento adiante. Hoje, quero agradecer por tudo que já fizeram, e dizer quanto são especiais para mim. Esta pessoa, cheia de alegria, forte e cheia de sonhos que me tornei foi graças a eles. Família é paz, é amor, é união.

Agradeço ao professor Dr João Paulo Papa, meu orientador deste projeto e da iniciação cientifica, por toda ajuda realizada durante a realização projeto desde o início. Sou muito a grato as oportunidades que o professor me ofereceu e por ter permitido entrar na área de Redes Neurais Artificias em que sou apaixonado.

Ao meu coorientador Gustavo Rosa que foi uma ajuda essencial para realizar meus sonhos. Um grande amigo e mestre, sempre com disposição para ajudar os outros. Ele conseguiu revisar todos os meus textos e ainda me encorajava aos estudos. Espero, trabalharmos mais vezes juntos.

Agradeço ao prof Dinesh e todo o grupo que me receberam muito bem na Austrália. Também sou grato ao RMIT University por ter me aceito. A todo conhecimento adquirido e suporte fornecido. Estudar no exterior enriqueceu meu potencial e habilidades, permitindo-me ver o mundo através de outras culturas e, espero, devolver à sociedade todo aprendizado absorvido nesta experiência.

Ao grupo de pesquisa Recogna, que também me receberam muito bem e me deram apoio. Fico feliz em fazer parte de um grupo unido e bastante reconhecido.

À FAPESP pelo suporte financeiro para o desenvolvimento de dois projetos: projeto de Iniciação Científica e Bolsa Estágio de Pesquisa no Exterior.

A todos os professores da UNESP e USP pelos conhecimentos transmitidos durante a graduação. Guardarei todos em minha memória.

A todos os amigos que fiz aqui em Bauru e São Paulo, pelo companheirismo, por me ajudarem sempre que precisei, e por todos os momentos vividos nestes cinco anos de graduação.

"Você pode encarar um erro como uma besteira a ser esquecida ou como um resultado que aponta uma nova direção."

(Steve Jobs)

Resumo

O presente trabalho foca em estudar a eficiência da aplicação de Programação Genética e Algoritmo Genético nos hiperparâmetros da Máquina de Bolztmann Restrita Convolucional. Para este projeto, é relevante dizer que foi selecionado apenas três hiperparâmetros para otimização; foram definidos quatro conjuntos de base e; definidos parâmetros específicos como a taxa de probabilidade de cruzamento ser igual a 40%. Além disso, o trabalho busca construir uma interface em que o usuário pode interagir selecionando os intervalos de todos os parâmetros das técnicas meta-heurísticas, hiperparâmetros da rede neural e o conjunto da base na qual será utilizado. A interface também gera gráficos de curva de aprendizagem e uma tabela que apresenta informações de cada geração. Por fim, são apresentadas conclusões com base nos resultados obtidos.

Palavras-chave: Meta-heurística, Máquina de Boltzmann Restrita Convolucional, Programação Genética, Algoritmo Genético, Otimização de Hiperparâmetros, Rede Neural Artificial, Inteligência Artificial.

Abstract

This work presents a meta-heuristic optimization approach to fine-tune the hyperparameters of a neural network. This work focuses on studying the application of Genetic Programming and Genetic Algorithm to select hyperparameters of a Convolutional Restricted Boltzmann Machine. For this project, it is relevant to say that just three hyperparameters were chosen for optimization and four datasets. Furthermore, the project aims to build a Graphical User Interface in which the user can interact, select the ranges of meta-heuristic arguments, hyperparameters of a neural network and the dataset which will be used. The graphical interface also generates plots showing the learning curve and a table which presents information about each generation. Finally, the conclusion states the experimental results and future works.

Keywords: Meta-heuristics, Convolutional Restricted Boltzmann Machine, Genetic Programming, Genetic Algorithm, Hyperparameter Optimization, Artificial Neural Network, Artificial Intelligence.

Lista de figuras

Lista de tabelas

Lista de abreviaturas e siglas

CNN	*Convolutional Neural Network*
CD	*Contrastive Divergence*
CRBM	*Convolutional Restricted Boltzmann Machine*
GA	*Genetic Algorithm*
GP	*Genetic Programming*
MCMC	*Markov Chain Monte Carlo*
PoE	*Product of Experts*
RBM	*Restricted Boltzmann Machine*
RMSE	*Root Mean Squared Error*
SGD	*Stochastic Gradient Descent*

Sumário

1 Introdução

Os aplicativos de aprendizado de máquina tendem a usar representações de alto nível ao invés de dados brutos em uma ampla variedade de tarefas, como classificação e análise de imagens médicas. Por exemplo, ao treinar um classificador para reconhecer imagens, os *pixels* geralmente são mapeados em características e, em seguida, classificados. A maioria das abordagens convencionais usadas na área de análise de imagens médicas, por exemplo, bancos de filtros Gaussianos e *wavelets* (NIXON; AGUADO, 2012), são pré-projetadas, ou seja, são métodos gerais não ajustados para problemas ou conjuntos de dados específicos.

Uma alternativa para superar o problema do filtro pré-projetado é o aprendizado de representação, que são métodos que aprendem representações diretamente dos dados. Normalmente, esta metodologia é derivada dos dados, a qual descarta informações irrelevantes e preserva apenas detalhes úteis que cumprem com o objetivo da tarefa. Como estas características são otimizadas para um problema específico, elas podem fornecer um resultado de classificação melhor do que a abordagem dos bancos de filtros pré-projetados.

A Máquina de Boltzmann Restrita (*Restricted Boltzmann Machine* - RBM) proposta por Hinton, Osindero e Teh (2006) é um algoritmo de aprendizado de máquina baseado em *feature learning* e pouco usado em tarefas de análise de imagens médicas (TULDER; BRUIJNE, 2014). As RBMs podem ser interpretadas como redes neurais estocásticas e, embora sejam usadas principalmente para reconstrução de imagens e filtragem colaborativa, trabalham diretamente nos dados de entrada (*pixels*). Além disso, são arquiteturas excelentes que podem lidar com o aprendizado não supervisionado.

No entanto, como mencionado anteriormente, a melhor abordagem ao lidar com a análise de imagens médicas é extrair características de alto nível das imagens de entrada. A RBM Convolucional (*Convolutional Restricted Boltzmann Machine* - CRBM) (LEE et al., 2009) surge na tentativa de suavizar esse problema usando uma metodologia de compartilhamento de pesos, reduzindo o número de pesos e adicionando uma invariância translacional ao modelo. Pode-se ver este procedimento como o núcleo de convolução da Rede Neural Convolucional (*Convolutional Neural Network* -CNN) (LECUN et al., 1998). Além disso, as CRBMs são modelos não supervisionados e podem ser treinados da mesma forma que as RBMs.

Outro intrigante problema relacionado às técnicas de aprendizado de máquina é a seleção de hiperparâmetros. Na maioria das vezes, os hiperparâmetros são empiricamente escolhidos ou aperfeiçoados manualmente, o que pode não ser apropriado para problemas específicos. Assim, a tarefa de seleção de modelos visa encontrar um conjunto adequado de hiperparâmetros que maximiza/minimiza uma função de aptidão, como a qualidade de agrupamento em tarefas não supervisionadas ou a precisão do classificador em problemas

supervisionados. Uma abordagem interessante para lidar com problemas de otimização são os algoritmos biologicamente inspirados, conhecidos por meta-heurísticas. Esses algoritmos fornecem soluções excelentes em uma ampla variedade de aplicações.

Este trabalho de conclusão de curso diz respeito ao problema de selecionar adequadamente os hiperparâmetros de CRBMs como uma tarefa de otimização orientada por meta-heurísticas, na qual agentes codificam os valores dos hiperparâmetros em um problema de busca guiado pela função de perda sobre o conjunto de validação. No que nos diz respeito, este é o primeiro trabalho que tenta abordar o problema de seleção adequada de hiperparâmetros em Máquinas de Bolztmann Restritas Convolucionais. Para validar a abordagem proposta, foi empregado a Programação Genética, do inglês *Genetic Programming* (GP) (KOZA, 1992) e o Algoritmo Genético (GA), do inglês *Genetic Algorithm* (GA) (KOZA, 1992), por serem técnicas de otimização meta-heurística bem conhecidas, consistentes e baseadas na Teoria da Evolução de Darwin. Na prática, GP e GA funcionam de maneira bastante similar, onde ambas técnicas modelam problemas de otimização em estruturas que permitam, posteriormente, proporcionar novas soluções por meio de operadores de cruzamento (*crossover*), mutação e reprodução.

O restante do trabalho está organizado como segue. A Seção 2 apresenta uma fundamentação teórica em Máquinas de Bolztaman Restritas, Máquinas de Bolztaman Restritas Convolucionais, Algoritmo Genético e Programação Genética. A metodologia da presente proposta é apresentada na Seção 3 e os resultados são apresentadas na Seção 4. Finalmente, a Seção 5 apresenta a conclusão deste trabalho e uma subseção em que sugere sugestões para trabalhos futuros. Após as referências, encontram se os apêndices. O Apêndice A trata sobre os cálculos das derivadas dos parâmetros da RBM, o Apêndice B discute a implementação e configurações utilizando o framework DEAP e o Apêndice C apresenta a interface gráfica desenvolvida neste trabalho.

2 Fundamentação Teórica

2.1 Máquinas de Boltzmann Restritas

Máquinas de Boltzmann Restritas são redes neurais estocásticas baseadas em energia. Elas tiveram um papel muito importante no ressurgimento das redes neurais em 2006, permitindo o treinamento eficiente de modelos profundos. Desenvolvido por Geoffrey Hinton (2006), a técnica *Contrastive Divergence* (CD) tornou o treinamento mais eficiente, pois o procedimento antigo consome um longo tempo para obter um resultado adequado.

Dentre as aplicações mais notáveis da RBM estão a redução de dimensionalidade (NOULAS; KRÖSE et al., 2008), pré-treinamento de redes neurais (PLAHL et al., 2012), detecção de fraudes (PUMSIRIRAT; YAN, 2018) e sistemas de recomendação colaborativos (SALAKHUTDINOV; MNIH; HINTON, 2007). Recentemente, esse tipo de rede neural foi usada pelos campeões do Prêmio Netflix (KOREN, 2009), uma competição em que a empresa pagaria 1 milhão de dólares para quem conseguisse melhorar o seu sistema de recomendação em 10%. Os competidores atingiram um alto desempenho em filtragem colaborativa e conseguiram superar a maior parte da concorrência. Com este acontecimento, RBMs ganharam grande popularidade.

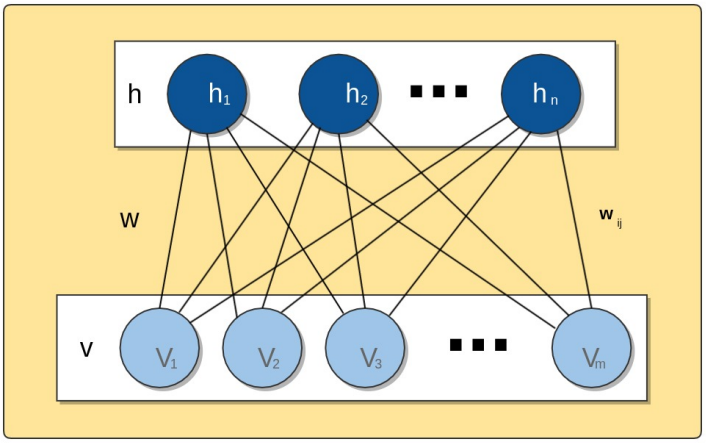
Originalmente, a RBM foi desenvolvida para dados binários, tanto na camada visível quanto na camada oculta. Essa abordagem é conhecida como Bernoulli-Bernoulli RBM (BBRBM). Devido ao fato de existir problemas onde é necessário processar outros tipos de dados, foi posteriormente proposta a Gaussian-Bernoulli RBM (GBRBM) (CHO; ILIN; RAIKO, 2011), que utiliza uma distribuição normal para modelar os neurônios da camada visível.

2.1.1 Arquitetura

As RBMs são compostas por duas camadas de neurônios (visíveis e ocultos), nas quais a fase de aprendizagem é conduzida de maneira não-supervisionada. As conexões entre neurônios são bidirecionais e simétricas, ou seja, existe tráfego de informação em ambos os sentidos da rede. Além disso, para simplificar os procedimentos de inferência, os neurônios de uma mesma camada não estão conectados entre si. Portanto, só existe conexão entre neurônios de camadas diferentes, por isso a máquina é restrita.

Uma arquitetura tradicional de uma Máquina de Boltzmann Restrita é descrita na Figura 1, que compreende uma camada visível $\mathbf{v}$ com m unidades e uma camada oculta $\mathbf{h}$ com n unidades. Adicionalmente, uma matriz de valores reais $\mathbf{W}_{m \times n}$ modela os pesos entre os neurônios visíveis e ocultos, em que w_{ij} representa o peso entre a unidade visível v_i e a unidade oculta h_j.

Figura 1 – Arquitetura de uma RBM.

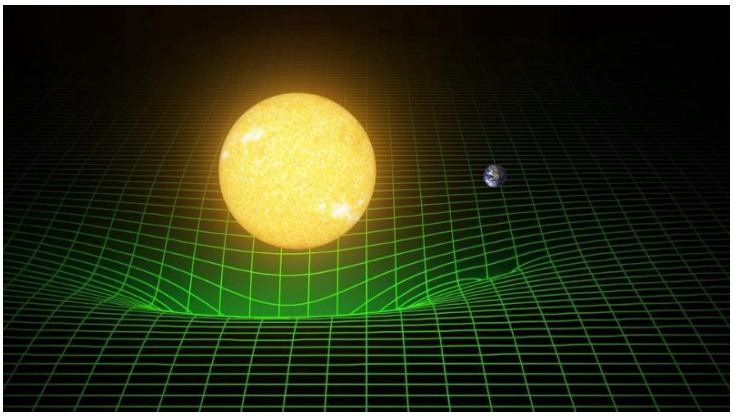


Fonte: Elaborada pelo autor.

2.1.2 Modelo Baseado em Energia

A energia é uma propriedade quantitativa da física que ajuda na investigação de fenômenos. Por exemplo, como evidenciado na Figura 2, a energia gravitacional descreve a energia potencial que um corpo com massa tem em relação a outro devido à gravidade. A partir do conceito, parece difícil associá-lo à aprendizado em profundidade inicialmente, no entanto existem arquiteturas que utilizam da ideia de energia como uma métrica para mensurar a qualidade dos modelos.

Figura 2 – Energia gravitacional de dois corpos.



Fonte: Siegel (2019)

Um dos propósitos dos modelos de energia é avaliar a dependência entre as variáveis. Essa relação ocorre por meio da associação de uma energia escalar à cada configuração das variáveis, que serve como medida de compatibilidade. Uma alta energia significa uma má compatibilidade. Um modelo baseado em energia tenta sempre minimizar uma função de energia predefinida.

As variáveis **v** e **h** são definidas como sendo unidades binárias, em outras palavras, $\mathbf{v} \in \{0,1\}^m$ e $\mathbf{h} \in \{0,1\}^n$. A função de energia de uma Máquina de Boltzmann Restrita de Bernoulli é dada por:

$$E(\mathbf{v},\mathbf{h}) = -\sum_{i=1}^{m} a_i v_i - \sum_{j=1}^{n} b_j h_j - \sum_{i=1}^{m}\sum_{j=1}^{n} v_i h_j w_{ij}, \tag{2.1}$$

em que **a** e **b** correspondem às unidades de bias das camadas visível e oculta, respectivamente.

Como pode se notar, o valor da função de energia depende das configurações de estados visíveis/entrada, estados ocultos, pesos e viés. O treinamento de RBM consiste em encontrar parâmetros para determinados valores de entrada, de modo que a energia atinja um mínimo.

2.1.3 Modelo Probabilístico

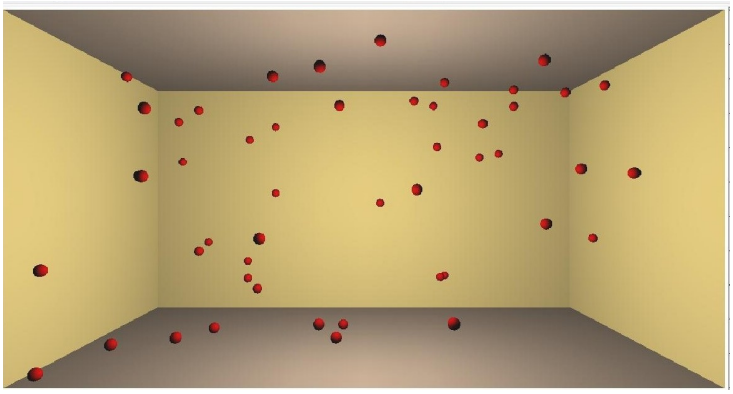
A RBM tem como inspiração a distribuição de Boltzmann (SANTORO, 2011), a qual tem origem na Termodinâmica, e atribui a um sistema único com temperatura e função de energia bem definidas, a probabilidade de estar em um determinado estado. Na mecânica estatística, a Distribuição de Boltzmann é dada por:

$$p_i = \frac{e^{-\epsilon_i/k\mathrm{T}}}{\sum_{j=1}^{M} e^{-\epsilon_j/k\mathrm{T}}}, \tag{2.2}$$

em que p_i é a probabilidade do estado i, ϵ_i a energia do estado i, k a constante de Boltzmann, T a temperatura do sistema e M é o número de estados possíveis do sistema. Devido à existência de diversos problemas que podem ser modelados pela distribuição de Boltzmann, o termo sistema tem um significado muito amplo, podendo variar de uma dimensão atômica à uma macroscópica, como um tanque de armazenamento de gás natural, por exemplo. A distribuição mostra que os estados com menor energia terão sempre maior probabilidade de serem ocupados, do que os estados com maior energia.

Para entender de forma intuitiva, segue um exemplo ilustrativo. Supondo que existe um certo gás dentro de uma sala a uma dada temperatura compondo um sistema. Este sistema possui diferentes estados possíveis, no qual o gás pode ocupar qualquer lugar da sala. A distribuição nos fornece que a maior probabilidade é quando o gás ocupa a sala inteira, como mostra a Figura 3 pois, devido à suas propriedades físico-químicas, as moléculas de gás tendem a ocupar todo o espaço disponível e não apenas um canto da sala, e este estado acontece quando o sistema assume menor energia.

Figura 3 – Moléculas de gás ocupando a sala.



Fonte: Siegel (2019)

Pode-se, analogamente, modelar essa distribuição em diversas situações. Outro caso seria um sistema composto por um copo, água e óleo. A distribuição irá mostrar que o evento de maior probabilidade e, consequentemente menor energia, é quando o óleo está em cima da água por conta de suas propriedades físico-químicas.

Em configurações matemáticas gerais, a distribuição de Boltzmann é também conhecida como *Gibbs measure*. Em aprendizado em profundidade, ela é usada na distribuição amostral de redes neurais estocásticas e possui a seguinte probabilidade de configuração $(\mathbf{v}, \mathbf{h})$ calculada como:

$$P(\mathbf{v}, \mathbf{h}) = \frac{1}{Z} e^{-E(\mathbf{v},\mathbf{h})} \tag{2.3}$$

$$Z = \sum_{\mathbf{v}\in\{0,1\}^m} \sum_{\mathbf{h}\in\{0,1\}^n} e^{-E(\mathbf{v},\mathbf{h})},$$

em que Z representa a chamada função de partição, que é um fator de normalização calculado sobre todas as possíveis configurações envolvendo as unidades visíveis e ocultas. Lembrando que as variáveis que consistem os vetores são do tipo Bernoulli, verifica-se que seu domínio é dado por $\{0,1\}^m \times \{0,1\}^n$, ou seja, formado por todas combinações possíveis de valores das variáveis visíveis e ocultas. Em cada ponto no tempo, a RBM está em um determinado estado que refere-se aos valores dos neurônios nas camadas visíveis e ocultas, respectivamente **v** e **h**.

A energia de uma RBM é uma função bilinear (linear em **v** e **h**) e, portanto, particularmente simples. Muitas vezes, é útil escrever a distribuição de probabilidade, Equação 2.3 em conjunto com a Equação 2.1, correspondente como:

$$P(\mathbf{v},\mathbf{h}) \overset{(2.3)}{=} \frac{1}{Z}\exp\left(-E(\mathbf{v},\mathbf{h})\right) \tag{2.4}$$

$$\overset{(2.1)}{=} \frac{1}{Z}\exp\left(\mathbf{v}^T\mathbf{a}+\mathbf{b}^T\mathbf{h}+\mathbf{v}^T\mathbf{w}\mathbf{h}\right) \tag{2.5}$$

$$= \frac{1}{Z}\exp\left(\mathbf{v}^T\mathbf{a}+\sum_{j=1}^{n}(b_j+\mathbf{v}^T\mathbf{w}_j)h_j\right) \tag{2.6}$$

$$= \frac{1}{Z}\exp\left(\mathbf{v}^T\mathbf{a}\right)\prod_{j=1}^{n}\exp\left((b_j+\mathbf{v}^T\mathbf{w}_j)h_j\right). \tag{2.7}$$

Assim como na Física, a distribuição de Boltzmann atribui uma probabilidade para observar um estado de **v** e **h**, que depende do total de energia do modelo. Infelizmente, é muito difícil calcular a probabilidade conjunta devido ao grande número de combinações possíveis de **v** e **h** na função de partição Z. As variáveis ocultas foram introduzidas para aumentar o poder de representação do modelo, elas não são de interesse direto.

2.1.4 Probabilidade Marginal

Em muitos casos de interesse, deseja-se encontrar a probabilidade das variáveis visíveis (**v**), ou seja, aquelas vistas externamente, independentemente das variáveis ocultas (**h**). Assim, a probabilidade marginal de um vetor visível (entrada) é dada por:

$$P(\mathbf{v}) = \sum_{\mathbf{h}\in\{0,1\}^n} P(\mathbf{v},\mathbf{h}) \tag{2.8}$$

$$\overset{(2.7)}{=} \sum_{\mathbf{h}}\frac{1}{Z}\exp\left(\mathbf{v}^T\mathbf{a}\right)\prod_{j=1}^{n}\exp\left((b_j+\mathbf{v}^T\mathbf{w}_j)h_j\right) \tag{2.9}$$

$$= \frac{1}{Z}\exp\left(\mathbf{v}^T\mathbf{a}\right)\sum_{h_1=0}^{1}\ldots\sum_{h_n=0}^{1}\prod_{j=1}^{n}\exp\left((b_j+\mathbf{v}^T\mathbf{w}_j)h_j\right) \tag{2.10}$$

$$= \frac{1}{Z}\exp\left(\mathbf{v}^T\mathbf{a}\right)\prod_{j=1}^{n}\left[\sum_{h_j=0}^{1}\exp\left((b_j+\mathbf{v}^T\mathbf{w}_j)h_j\right)\right] \tag{2.11}$$

$$= \frac{1}{Z}\exp\left(\mathbf{v}^T\mathbf{a}\right)\prod_{j=1}^{n}\left(1+\exp(b_j+\mathbf{v}^T\mathbf{w}_j)\right) \text{ (desde que } h_j\in\{0,1\}) \tag{2.12}$$

O passo (8) é análogo à $a_1b_1c_1+a_1b_1c_2+a_1b_2c_1+a_2b_1c_1+a_1b_2c_2+a_2b_1c_2+a_2b_2c_1+a_2b_2c_2 = (a_1+a_2)(b_1+b_2)(c_1+c_2)$ e economiza o poder computacional. Portanto, $P(\mathbf{v})$ pode ser calculado de forma bastante eficiente.

Conforme a energia definida pela Equação 2.3, é possível obter outra definição de $P(\mathbf{v})$ definida da seguinte forma:

$$P(\mathbf{v}) = \frac{1}{Z}\exp(-\mathcal{F}(\mathbf{v})), \tag{2.13}$$

em que

$$Z = \sum_{\mathbf{v} \in \{0,1\}^m} \exp(-\mathcal{F}(\mathbf{v})). \tag{2.14}$$

A função $\mathcal{F}(\mathbf{v})$ é denominada *free energy*, inspirada da Física, calculada como segue:

$$\begin{aligned}
\frac{1}{Z}\exp(-\mathcal{F}(\mathbf{v})) &\overset{(2.13)}{=} P(\mathbf{v}) && (2.15)\\
&= \sum_{\mathbf{h}} P(\mathbf{v}, \mathbf{h}) && (2.16)\\
&\overset{(2.3)}{=} \sum_{\mathbf{h}} \frac{1}{Z}\exp\left(-E(\mathbf{v}, \mathbf{h})\right) && (2.17)\\
\iff \mathcal{F}(\mathbf{v}) &= -\ln\left[\sum_{\mathbf{h} \in \{0,1\}^n} \exp\left(-E(\mathbf{v}, \mathbf{h})\right)\right]. && (2.18)
\end{aligned}$$

2.1.5 Probabilidade Condicional

Devido à sua importância no contexto das RBMs, as distribuições condicionais $P(\mathbf{v}|\mathbf{h})$ e $P(\mathbf{h}|\mathbf{v})$ serão apresentadas aqui. A estrutura bipartida do modelo faz com que suas distribuições condicionais sejam calculadas de forma eficiente, como segue:

$$\begin{aligned}
P(\mathbf{h}|\mathbf{v}) &= \frac{P(\mathbf{v}, \mathbf{h})}{P(\mathbf{v})} && (2.19)\\
&\overset{(2.7,2.12)}{=} \frac{\prod_j \exp\left((b_j + \mathbf{v}^T\mathbf{w}_j)h_j\right)}{\prod_j \left(1 + \exp(b_j + \mathbf{v}^T\mathbf{w}_j)\right)} && (2.20)\\
&= \prod_j \frac{\exp\left((b_j + \mathbf{v}^T\mathbf{w}_j)h_j\right)}{1 + \exp(b_j + \mathbf{v}^T\mathbf{w}_j)} && (2.21)\\
&\overset{def.}{=} \prod_j P(h_j|\mathbf{v}). && (2.22)
\end{aligned}$$

A Equação 2.22 confirma que as variáveis ocultas são condicionalmente independentes dadas as variáveis visíveis. Conclui-se também, que a probabilidade de ativação de uma unidade visível dado $\mathbf{v}$, $P(h_j|\mathbf{v})$, é calculada como segue

$$\begin{aligned}
P(h_j = 1|\mathbf{v}) &\overset{(2.22)}{=} \frac{\exp\left(b_j + \mathbf{v}^T\mathbf{w}_j\right)}{1 + \exp(b_j + \mathbf{v}^T\mathbf{w}_j)} && (2.23)\\
&=: \sigma\left(b_j + \mathbf{v}^T\mathbf{w}_j\right) && (2.24)\\
&= y_j(\mathbf{v}), && (2.25)
\end{aligned}$$

em que $\sigma(m) = \frac{e^m}{1+e^m}$ é uma função sigmoidal padrão e y_j representa a função de ativação de um neurônio conhecida em redes neurais artificiais. Além disso, os termos de bias b_j e os vetores de pesos sinápticos w_j pertencem a redes neurais artificias.

Devido à simetria da energia **v** e **h**, expressões análogas podem ser derivadas para $P(\mathbf{v}|\mathbf{h})$ e $P(v_i = 1|\mathbf{h})$. Assim, a rede neural também pode ser executada de trás para frente, mas com termos de bias a_i e os vetores de linha da matriz **w** como vetores de peso, de modo que os pesos da etapa *feedback* sejam os mesmos que os da etapa *feed-forward*.

2.1.6 Maximizando a probabilidade

No treinamento de RBMs, considera-se uma amostra $v^{(t)}$ extraída da distribuição empírica dos dados de treinamento, e objetiva-se determinar os parâmetros do modelo que minimizem a função de logaritmo-verossimilhança negativa, definida como:

$$\mathcal{L}(\theta; v^{(t)}) = -\ln P(v^{(t)}), \tag{2.26}$$

sendo que $v^{(t)}$ é uma instância de **v** no tempo t e $\theta = [W, a, b]$ é um vetor em que cada componente consiste em um dos parâmetros do modelo da RBM. $\mathcal{L}(\theta; v^{(t)})$ representa a função perda (ou custo) a ser minimizada pelo método método do gradiente descendente estocástico (SGD). Em razão do sinal negativo na Equação 2.26, percebe-se que a minimização de $\mathcal{L}(\theta; v^{(t)})$ é equivalente à maximização do logaritmo-verossimilhança da amostra de treinamento, um objetivo de otimização que é comumente empregado na estimação de parâmetros de modelos estatísticos.

2.1.7 O gradiente do modelo

Para simplificar um pouco a escrita, diferencia-se a Equação 2.26 com auxilio da Equação 2.13, como representado abaixo:

$$-\ln(P(\mathbf{v}^{(t)})) \overset{(2.13)}{=} \mathcal{F}(\mathbf{v}^{(t)}) + \ln(Z). \tag{2.27}$$

Tendo que $Z = \sum_{\mathbf{v}\in\{0,1\}^m} \exp(-\mathcal{F}(\mathbf{v}))$, o gradiente é calculado de uma forma geral:

$$-\frac{\partial \ln(P(\mathbf{v}^{(t)}))}{\partial\theta} = \frac{\partial\mathcal{F}(\mathbf{v}^{(t)})}{\partial\theta} + \frac{1}{Z}\frac{\partial Z}{\partial\theta} \tag{2.28}$$

$$= \frac{\partial\mathcal{F}(\mathbf{v}^{(t)})}{\partial\theta} - \frac{1}{Z}\sum_{\mathbf{v}\in\{0,1\}^m} \exp(-\mathcal{F}(\mathbf{v})) \cdot \frac{\partial\mathcal{F}(\mathbf{v})}{\partial\theta} \tag{2.29}$$

$$= \frac{\partial\mathcal{F}(\mathbf{v}^{(t)})}{\partial\theta} - \sum_{\mathbf{v}\in\{0,1\}^m} P(\mathbf{v}) \cdot \frac{\partial\mathcal{F}(\mathbf{v})}{\partial\theta} \tag{2.30}$$

O primeiro termo dessa derivada é chamado de fase positiva, pois sua função é aumentar a probabilidade dos dados. Ele é semelhante à média da derivada da energia quando as amostras dos dados estão acopladas no lugar das unidades visíveis. O segundo termo é denominado de fase negativa, pois sua função é reduzir a probabilidade de amostras geradas pelo modelo. Ele

atua como a média da derivada da energia quando não há amostras acopladas no lugar das unidades visíveis.

O problema a seguir, é calcular o termo negativo. Ele é simplesmente a expectativa de todas as configurações possíveis dos dados **v** sob a distribuição do modelo. Como isso costuma ser maior que o número estimado de átomos no universo, convém ter alguma estimativa para computar o termo negativo.

2.1.8 Amostragem sobre a distribuição do modelo

Como geralmente há muitos estados de v, não se pode somar todas as probabilidades sobre eles. Então, o primeiro passo para tornar esse cálculo tratável é estimar a expectativa usando um número fixo de amostras do modelo ao invés de utilizar toda a distribuição.

As amostras usadas para estimar o gradiente da fase negativa são referidas como partículas negativas, que são denotadas como $\mathcal{N}$. O gradiente pode então ser escrito como:

$$-\frac{\partial \ln P(v)}{\partial \theta} \approx \frac{\partial F(v)}{\partial \theta} - \frac{1}{|\mathcal{N}|} \sum_{\tilde{v} \in \mathcal{N}} \frac{\partial F(\tilde{v})}{\partial \theta}, \tag{2.31}$$

em que $\tilde{v}$ é uma unidade amostral pertencente ao conjunto $\mathcal{N}$. Com a fórmula dada pela Equação 2.31, quase obtém-se um algoritmo prático e estocástico de aprendizado para um modelo baseado em energia. O único ingrediente ausente é como extrair essas partículas negativas $\mathcal{N}$, ou seja, é necessário saber como conseguir as amostras.

Na estatística, o processo de selecionar um grupo de indivíduos (amostra) de uma população é chamado de amostragem. A literatura estatística é rica em métodos de amostragem e o método de Monte Carlo via cadeias de Markov (MCMC), conhecido como amostragem de *Gibbs*, é especialmente adequado para situações em que amostrar a partir das distribuições condicionais do modelo seja simples, como ocorre com as RBMs.

A amostragem de Gibbs da junção de N variáveis aleatórias $S = (S_1, ..., S_N)$ é dado através de uma sequência de N sub-passos de amostragem da forma $S_i \sim p(S_i|S_{-i})$ onde S_{-i} contém as $N-1$ outras variáveis aleatórias de S excluindo S_i.

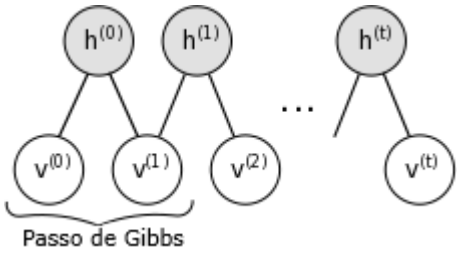
Para RBMs, existe um fator simplificador porque as variáveis ocultas são conjuntamente independentes dados os estados das visíveis. Essa configuração possibilita que as unidades ocultas sejam amostradas simultaneamente, segundo a distribuição $P(h, v)$. De forma análoga, vale para as unidades visíveis. Então, um passo de amostragem de Gibbs pode ser escrito com apenas dois sub-passos:

- Produz-se $h^{(n)}$ segundo $P(h|V = v^{(n-1)})$;
- E, produz-se $v^{(n)}$ segundo $P(v|H = h^{(n)})$.

O índice n sobrescrito aos nomes das amostras h e v indica o passo de Gibbs em que as mesmas seriam produzidas. A produção de $h^{(n)}$ consiste no simples sorteio de amostras independentes com distribuição de Bernoulli cada qual tendo uma probabilidade de sucesso igual a $P(h_j = 1|V = v^{(n-1)})$, para j = 1,...,n_h. A produção de $v^{(v)}$, é feita de forma análoga.

A Figura 4 ilustra a sequência ou cadeia de amostragens definida pelos passos e sub-passos do amostrador de Gibbs. Nota-se que a sequência de conexões $v^{(0)} \rightarrow h^{(1)} \rightarrow v^{(1)}$ representa o primeiro passo completo do amostrador. Após t iterações, um outro vetor $v^{(t)}$ é recriado a partir dos valores de entrada originais $v^{(0)}$.

Figura 4 – Representação da cadeia de amostragem de Gibbs em que $v^{(n)}$ e $h^{(n)}$ representa o conjunto de todas os nós visíveis e ocultos da n-ésima etapa da cadeia de Markov, respectivamente.



Fonte: Theano (2019)

Em teoria, cada atualização de parâmetro no processo de aprendizagem exigiria a execução de uma dessas cadeias para a convergência. Para evitar um custo computacional caro, vários algoritmos foram desenvolvidos para RBMs, a fim de amostrar eficientemente de $p(v, h)$ durante o processo de aprendizagem.

2.1.9 Divergência Contrastiva

No trabalho que Hinton introduziu o modelo denominado Produto de Especialistas (PoE) (HINTON, 2002), ele propôs o algoritmo CD tendo como objetivo o treinamento de PoEs. Além disso, Hinton mostrou que o algoritmo seria aplicável ao treinamento de RBMs, visto que uma RBM pode ser entendida como PoE.

A derivação do algoritmo CD inicia obtendo uma expressão tratável, que consiste em estimar o gradiente da fase negativa da Equação 2.30 usando-se uma única amostra. Então, tem-se agora a versão aproximada do gradiente da função perda:

$$\frac{\partial \mathcal{L}(\theta; v)}{\partial \theta} \approx \frac{\partial \mathcal{F}(v)}{\partial \theta} - \frac{\partial \mathcal{F}(\tilde{v})}{\partial \theta}, \tag{2.32}$$

sendo $\tilde{v}$ uma amostra obtida via amostragem de Gibbs e representativa da distribuição marginal das variáveis visíveis. A aproximação define que na amostragem de Gibbs sejam usados k passos

e que a cadeia de amostragem seja iniciada na amostra de treinamento, $v^{(t)}$. Nessas condições, denota-se o algoritmo por CD-*k*.

Segundo o método do gradiente descendente estocástico, a regra de atualização de parâmetros pode ser escrita como

$$\theta \leftarrow \theta - \eta \frac{\partial \mathcal{L}(\theta; v)}{\partial \theta}, \tag{2.33}$$

de modo que o uso de (2.32) nessa última expressão produz

$$\theta \leftarrow \theta - \eta \left(\frac{\partial \mathcal{F}(v)}{\partial \theta} - \frac{\partial \mathcal{F}(\tilde{v})}{\partial \theta} \right), \tag{2.34}$$

em que η é uma constante positiva chamada de taxa de aprendizagem. Essa expressão é desenvolvida para cada um dos parâmetros do modelo, usando as seguintes derivativas sobre a matriz de pesos $\mathbf{W}$, e vieses $\mathbf{a}$ e $\mathbf{b}$ na iteração t e chegando-se, assim, ao conjunto de regras de atualização a ser usado no algoritmo de treinamento:

$$\mathbf{W}^{t+1} = \mathbf{W}^t + \underbrace{\eta(P(\mathbf{h}=1|\mathbf{v})\mathbf{v}^T - P(\tilde{\mathbf{h}}=1|\tilde{\mathbf{v}})\tilde{\mathbf{v}}^T) + \Phi}_{=\Delta\mathbf{W}^t}, \tag{2.35}$$

$$\mathbf{a}^{t+1} = \mathbf{a}^t + \underbrace{\eta(\mathbf{v}-\tilde{\mathbf{v}}) + \alpha\Delta\mathbf{a}^{t-1}}_{=\Delta\mathbf{a}^t} \tag{2.36}$$

e

$$\mathbf{b}^{t+1} = \mathbf{b}^t + \underbrace{\eta(P(\mathbf{h}=1|\mathbf{v}) - P(\tilde{\mathbf{h}}=1|\tilde{\mathbf{v}})) + \alpha\Delta\mathbf{b}^{t-1}}_{=\Delta\mathbf{b}^t}, \tag{2.37}$$

onde λ e α denotam o peso de decaimento e o momento, respectivamente. De maneira geral, as Equações 2.35, 2.36 e 2.37 empregam o conhecido algoritmo de otimização de Gradiente Descendente. O termo adicional Φ na Equação 2.35 é usado para controlar os valores da matriz $\mathbf{W}$ durante o processo de convergência, e é formulado como segue:

$$\Phi = -\lambda\mathbf{W}^t + \alpha\Delta\mathbf{W}^{t-1}. \tag{2.38}$$

Uma versão em pseudo-código do CD-*k* pode ser vista no Algoritmo 1. A cadeia de Gibbs é inicializada com um exemplo de treinamento $v^{(0)}$ do conjunto de treino e produz a amostra $v^{(k)}$ após *k* passos. Cada etapa t consiste da amostragem $h^{(t)}$ de $p(h|v^{(t)})$ e amostra-

gem $v^{(t+1)}$ de $p(v|h^{(t)})$.

Algoritmo 1: Divergência Contrastiva k-passos

Entrada: RBM $(V_1, \ldots, V_m, H_1, \ldots, H_n)$, batch S de treinamento, η, α, λ

Saída: aproximação do gradiente $\Delta w_{ij}, \Delta a_j$ e Δb_i para $i = 1, \ldots, n,\ j = 1, \ldots, m$

inicia $\Delta w_{i,j} = \Delta a_j = \Delta b_i = 0$ para $i = 1, \ldots, n,\ j = 1, \ldots, m$

para todo $v \in S$ **faça**

  $v^{(0)} \leftarrow v$

  para $t = 0, \ldots, k-1$ **faça**

    para *i=1,...,n* **faça** amostra $h_i^{(t)} \sim p(h_i|v^{(t)})$;

    para *j=1,...,m* **faça** amostra $v_j^{(t+1)} \sim p(v_j|h^{(t)})$;

  fim

  para $i = 1, \ldots, n,\ j = 1, \ldots, m$ **faça**

    $\Delta w_{ij} \leftarrow \alpha \Delta w_{ij} + \eta(p(H_i = 1|v^{(0)}) \cdot v_j^{(0)} - p(H_i = 1|v^{(k)}) \cdot v_j^{(k)}) + \Phi$

    $\Delta a_j \leftarrow \alpha \Delta a_j + \eta(v_j^{(0)} - v_j^{(k)})$

    $\Delta b_i \leftarrow \alpha \Delta b_i + \eta(p(H_i = 1|v^{(0)}) - p(H_i = 1|v^{(k)}))$

  fim

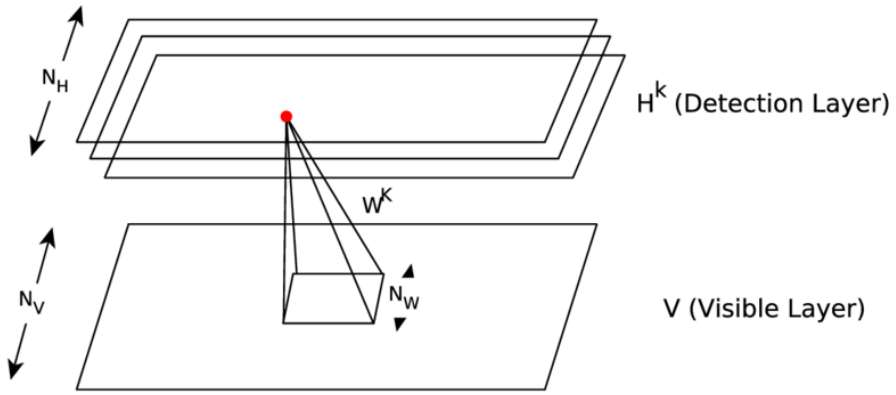
fim

2.2 Máquinas de Boltzmann Restritas Convolucionais

As Máquinas Boltzmann Restritas (HINTON, 2012) possuem arquiteturas ideais para trabalhar com a modelagem de dados. No entanto, elas são incapazes de funcionar adequadamente com a estrutura bidimensional das imagens, pois aprendem pesos separados para cada característica, tornando extremamente difícil dimensionar modelos em imagens completas. Uma abordagem recente que tenta superar esse problema é a Máquina Boltzmann Restrita Convolucional, no qual usa uma metodologia de compartilhamento de pesos da Rede Neural Convolucional.

Basicamente, a arquitetura da CRBM é semelhante a arquitetura de uma RBM. A única diferença é que os pesos da CRBM são compartilhados entre todos os locais das imagens. Em um modelo CRBM, as conexões compartilham pesos de forma semelhante ao *kernel* Convolucional e cada imagem resultante de um filtro é um mapa de características $\mathbf{H}$. A camada de entrada consiste em uma matriz $N_V \times N_V$, enquanto que a camada oculta consiste em K "grupos", onde cada grupo é uma matriz $N_H \times N_H$ de unidades binárias e tem associado um filtro $N_W \times N_W$ ($N_W = N_V - N_H + 1$). O resultado é $K \cdot N_{H^2}$ unidades ocultas e $K \cdot N_{W^2}$ unidades de filtros. Além disso, cada grupo oculto também possui um viés b_k, porém as unidades visíveis compartilham um único viés c.

Figura 5 – Arquitetura de uma CRBM. Os nós ocultos são agrupados em mapas de características $\mathbf{H}^1, \ldots, \mathbf{H}^k$ e os pesos das conexões são compartilhados utilizando os filtros convolucionais $\mathbf{W}_1, \ldots, \mathbf{W}_K$.



Fonte: Wicht, Fischer e Hennebert (2016).

A função de energia $E(\mathbf{v}, \mathbf{h})$ é definido como:

$$P(\mathbf{v},\mathbf{h}) = \frac{1}{Z}\exp(-E(\mathbf{v},\mathbf{h})), \tag{2.39}$$

$$E(\mathbf{v},\mathbf{h}) = -\sum_{k=1}^{K}\sum_{i,j=1}^{N_H}\sum_{r,s=1}^{N_W} h_{ij}^{k} W_{rs}^{k} v_{i+r-1,j+s-1} \tag{2.40}$$

$$-\sum_{k=1}^{k} b_k \sum_{i,j=1}^{N_H} h_{ij}^{k} - c\sum_{i,j=1} v_{ij}. \tag{2.41}$$

Conforme definido pelo algoritmo de aprendizado da RBM, também é possível executar o procedimento de amostragem de Gibbs para treinar as CRBMs. Seja $*$ o operador convolucional, pode-se definir as probabilidades condicionais da seguinte maneira:

$$P(h_{ij}^{k} = 1|\mathbf{v}) = \phi((\tilde{W}_k * v)_{ij} + b_k), \tag{2.42}$$

$$P(v_{ij} = 1|\mathbf{h}) = \phi((\sum_{k} W_k * h_k)_{ij} + c), \tag{2.43}$$

onde ϕ é a função sigmóide, $\tilde{W}_k$ é o filtro invertido horizontal e verticalmente de W_k, e $\cdot_{ij}$ indica o pixel no local (i, j).

Observe que, quando é realizado a primeira etapa da amostragem, calcular $P(h_{ij}^{k} = 1|\mathbf{v})$, a convolução encolhe a imagem. Ao realizar a volta do procedimento e tornar possível a amostragem que calcula $P(v_{ij} = 1|\mathbf{h})$, é necessário utilizar o *padding*. A técnica consiste em adicionar uma nova camada à borda de uma imagem e assim, a matriz resultante é do mesmo tamanho da matriz de entrada.

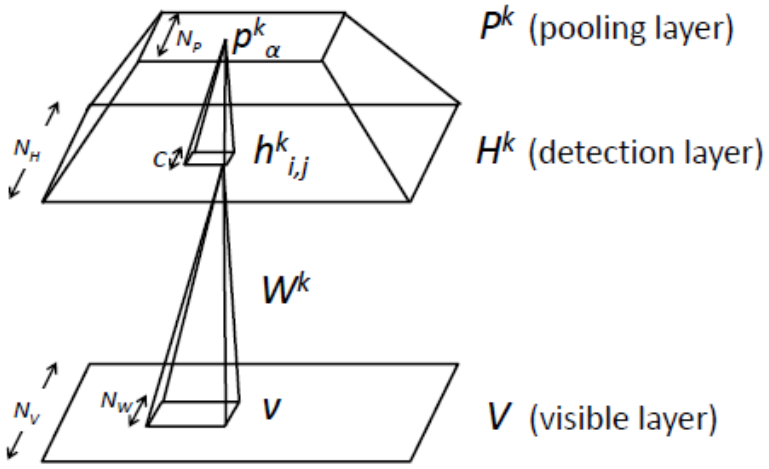
2.2.1 Max-pooling probabilístico

Em geral, os detectores de nível superior utilizam informações abstratas em aprendizado profundo. Existir representações invariantes à translação, como redes em convolucionais, geralmente envolvem dois tipos de camadas: camadas de "detecção", cujo resultado é calculado pela convolução de um filtro com a camada anterior, e camadas de "*pooling*", que diminuem a representação das camadas de detecção por um fator constante. Mais especificamente, cada unidade em uma camada de *pooling* calcula a ativação máxima das unidades de uma pequena região. Utilizá-lo permite que as representações da camada superior sejam invariantes a pequenas translações e reduz o custo computacional.

Honglak Lee, Roger Grosse, Rajesh Ranganath e Andrew Y. Ng (LEE et al., 2009) desenvolveram um novo modelo generativo para que a inferência envolva um comportamento semelhante ao de um *max-pooling*. Um modelo generativo de imagens que suporta inferência nos sentidos de cima para baixo e de baixo para cima. Para simplificar a notação, considera-se um modelo com uma camada visível V, uma camada de detecção H, e uma camada de

pooling P, como mostra a Figura 6. As camadas de detecção e *pooling* contém K grupos de unidades, e cada grupo da camada de *pooling* possui $N_P \times N_P$ unidades binárias. Para cada $k \in \{1, \cdots, K\}$, a camada de *pooling* P^k reduz a representação da camada de detecção H^k por um fator C ao longo de cada dimensão, onde C é um número inteiro e pequeno, como 2 ou 3. Ou seja, a camada de detecção H^k é particionada em blocos de tamanho $C \times C$, e cada bloco α é ligado a exatamente uma unidade binária p^k_α na camada de *pooling* (i.e., $N_P = N_H/C$). Formalmente, define-se $B_\alpha = \{(i,j) : h_{i,j}$ pertence ao bloco $\alpha\}$.

Figura 6 – Arquitetura de uma CRBM com *max-pooling*.



Fonte: Lee et al. (2009).

As unidades de detecção do bloco B_α e a unidade de *pooling* p_α estão conectadas em um único potencial que impõe as seguintes restrições: no máximo uma das unidades de detecção pode estar ativada, e a unidade de *pooling* está ativada se, e somente se, uma unidade de detecção estiver ativada. Equivalentemente, considera-se as $C^2 + 1$ unidades como uma única variável aleatória que pode assumir um dos possíveis valores de $C^2 + 1$: um valor para cada uma das unidades de detecção ativada e um valor indicando que todas as unidades estão desligadas.

Formalmente, a função de energia é definida da seguinte forma:

$$E(\mathbf{v}, \mathbf{h}) = -\sum_k \sum_{i,j} (h^k_{i,j} (\tilde{W}^k * v)_{i,j} + b_k h^k_{i,j}) - c \sum_{i,j} v_{i,j}$$
$$\text{subj. to} \sum_{(i,j) \in B_\alpha} h^k_{i,j} \leq 1, \forall k, \alpha.$$

A amostragem da camada de detecção H e da camada de *pooling* P, dado a camada

visível V, é discutido agora. O grupo k recebe o seguinte sinal *bottom-up* da camada V:

$$I(h_{i,j}^k) \triangleq b_k + (\tilde{W}^k * v)_{i,j}. \tag{2.44}$$

Cada bloco é amostrado independentemente como uma função multinomial de suas entradas. Suponha que $h_{i,j}^k$ é uma unidade oculta contida no bloco α (i.e, $(i,j) \in B_\alpha$), o aumento de energia causado pela ativação da unidade $h_{i,j}^k$ é $-I(h_{i,j}^k)$, e a probabilidade condicional é dada por:

$$P(h_{i,j}^k = 1|\mathbf{v}) = \frac{\exp(I(h_{i,j}^k))}{1 + \sum_{(i',j')\in B_\alpha} \exp(I(h_{i',j'}^k))}$$
$$P(p_\alpha^k = 0|\mathbf{v}) = \frac{1}{1 + \sum_{(i',j')\in B_\alpha} \exp(I(h_{i',j'}^k))}$$

Amostragem da camada visível V, dado a camada oculta H pode ser executado da mesma maneira descrita na Equação 2.43.

2.3 Algoritmo Genético

John Holland introduziu algoritmos genéticos (AGs) em 1960, são técnicas meta-heurísticas inspiradas na teoria da evolução natural de Charles Darwin. Pertencendo a grande classe de algoritmos evolutivos, são usados para achar soluções quase ótimas em problemas de busca e otimização. Possuem operadores bio-inspirados como hereditariedade, mutação, seleção natural e recombinação (*crossover*). Cabe notar que AGs são uma classe de procedimentos com muitos passos separados, e cada uma destes passos possuem muitas variações possíveis. Holland foi gradualmente refinando suas ideias e, em 1992 publicou o seu livro *Adaptation in Natural and Artificial Systems* (HOLLAND et al., 1992). Desde então, estes algoritmos vêm sendo aplicados com sucesso nos mais diversos problemas de otimização e aprendizado de máquina.

Como é visto a seguir, estes algoritmos simulam o processo de seleção natural, conceito básico da evolução genética biológica, onde os indivíduos mais aptos são selecionados para reprodução, a fim de produzir descendentes para a próxima geração.

2.3.1 Seleção Natural

O processo de seleção natural começa com a seleção de indivíduos mais aptos de uma população. Eles produzem descendentes que herdam as características dos pais e serão adicionados à próxima geração. Este processo continua iterando e, ao final, uma geração com os indivíduos mais aptos será produzida. Essa noção pode ser aplicada para um problema de pesquisa. Dado um conjunto de soluções para um problema e selecionado o melhor, cinco fases são consideradas em um algoritmo genético:

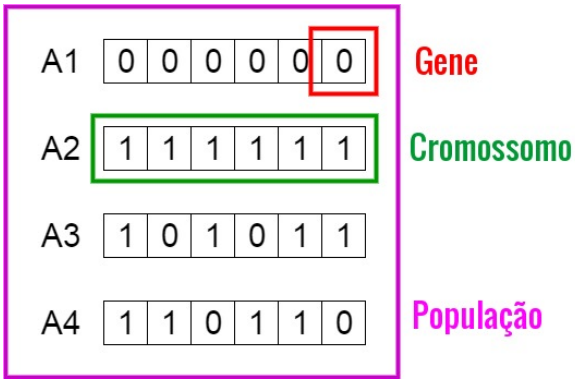
1. População inicial;
2. Função Objetiva;
3. Seleção;
4. Recombinação;
5. Mutação.

2.3.2 População Inicial

O processo começa com um conjunto de indivíduos chamado de população, e cada indivíduo é uma solução para o problema que deseja-se resolver. Um indivíduo é caracterizado por um conjunto de parâmetros que são conhecidos como genes. Serão os hiperarâmetros da CRBM nesse projeto.

Em um algoritmo genético, o conjunto de genes de um indivíduo é codificado em valores binários (*string* de 1s e 0s) e são unidos em uma *string* para formar um cromossomo (solução). Na Figura 7, o quadrado vermelho codifica um gene. Caso um gene necessite mais de dois valores, é necessário mais índices para codifica-lo. A literatura nos fornece várias formas de codificar um número inteiro ou real em valores binários.

Figura 7 – Representação da população, cromossomo e gene.



Fonte: Mallawaarachchi (2017).

.

O código genético deve ser capaz de representar todo o conjunto dos valores no espaço de busca e precisa ter tamanho finito.

2.3.3 Função Objetivo

A função objetivo determina a adequação de um indivíduo, ou seja, a capacidade de um indivíduo de competir com outros. Para isso, é dado uma pontuação de aptidão para cada indivíduo. A probabilidade de um indivíduo ser selecionado para reprodução baseia-se no seu índice de aptidão atribuído.

2.3.4 Seleção

A ideia da fase de seleção é encontrar os indivíduos mais aptos e deixá-los passar seus genes para a próxima geração. Dois pares de indivíduos são selecionados com base em suas notas de aptidão e utilizados para reproduzir, geralmente dois filhos. Existem diversos algoritmos de seleção, em geral, usa-se a técnica por “roleta", onde os indivíduos são ordenados em ordem decrescente e lhes são atribuídas aptidões acumuladas, representando sua faixa de escolha. É gerado então, um número aleatório entre 0 e a última aptidão acumulada, e o indivíduo selecionado é o primeiro com aptidão acumulada maior que o número aleatório gerado. Não funciona para valores negativos da função objetivo.

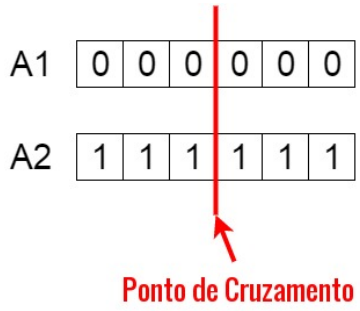
Podem ser aplicadas, ainda, outras formas de seleção para o problema a ser tratado. Como exemplos pode-se citar a seleção por “torneio", onde n indivíduos são escolhidos com

a mesma probabilidade, sendo selecionado o indivíduo com maior aptidão. Após, repete-se esse processo até preencher a população desejada. A seleção por “classificação"ou “ranking", semelhante à seleção por “roleta", com a diferença de que a probabilidade de seleção é relacionada à sua posição na ordenação dos indivíduos da população e não à sua aptidão em si. Finalmente, a seleção por “truncamento", onde são selecionados os n melhores indivíduos da população, descartando-se os outros. A maioria dos métodos de seleção são projetados para escolher preferencialmente indivíduos com maiores notas de aptidão, embora não exclusivamente, a fim de manter a diversidade da população.

2.3.5 Recombinação

Recombinação é a fase mais significativa em um Algoritmo Genético, visto que é responsável pela combinação de características dos pais para permitir que as próximas gerações as herdem. Para cada par de pais, um ponto de cruzamento é escolhido aleatoriamente e a partir deste ponto as informações serão trocadas. Por exemplo, considere o ponto de cruzamento como 3, conforme mostrado na Figura 8.

Figura 8 – Ponto de cruzamento.

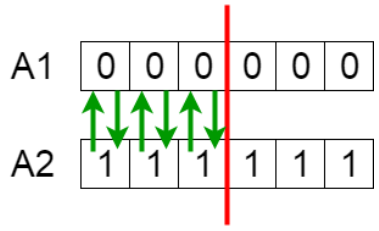


Fonte: Mallawaarachchi (2017)

.

Os descendentes são criados trocando os genes dos pais entre si até que o ponto de cruzamento seja alcançado, conforme a Figura 9.

Figura 9 – Troca de genes entre os pais.



Fonte: Mallawaarachchi (2017)

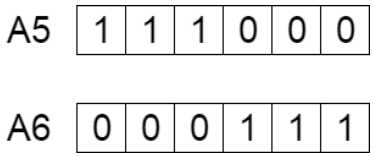
.

É possível, também, utilizar esse operador de outras maneiras, como:

- multi-pontos: é uma generalização da ideia de troca de material genético, onde muitos pontos de cruzamento podem ser utilizados.
- uniforme: A partir de uma máscara de *bits* aleatórios, é combinado os *bits* dos pais para gerar o filho.

Após serem gerados, conforme Figura 10, e antes de serem adicionados à população, os filhos passarão pelo processo de mutação.

Figura 10 – Resultado da Recombinação.



Fonte: Mallawaarachchi (2017)

.

2.3.6 Mutação

A mutação é um operador genético usado para manter a diversidade dentro da população, impedindo que a busca fique estagnada em um mínimo local e prevenindo a convergência prematura. Ela altera um ou mais valores genéticos em um cromossomo a partir de seu estado inicial. Os genes dos filhos gerados da recombinação são submetidos a uma mutação com baixa probabilidade definida pelo usuário. Para cada gene, é retirado um número aleatório no intervalo $[0, 1]$ e comparado com a taxa de mutação. Caso esse valor seja menor que a taxa, então o gene é alterado.

Devido a existirem diversas codificações de genes, existem diversos tipos de mutação como *Boundary*, *Non-Uniform*, *Uniform*, *Gaussian*, *Shrink* e *Flip Bit*. Para o nosso caso, utilizamos o *Flip Bit*, no qual basta inverter o bit. A Figura 11 mostra um exemplo dessa mutação.

Figura 11 – Exemplo de uma mutação.

Fonte: Mallawaarachchi (2017).

.

2.3.7 Critérios de parada

O processo de gerações é repetido até que um critério de parada seja atingido. Alguns critérios comuns são:

- Uma solução é encontrada que satisfaz os critérios mínimos do problema;
- O número fixo de gerações foi alcançada;
- Recurso alocado (tempo de computação / dinheiro) atingido;
- A população convergiu, não produzindo assim, descendentes significativamente diferentes da geração anterior;
- Inspeção manual;
- Combinações dos itens anteriores.

Segue um exemplo de algoritmo genético:

Algoritmo 2: Algoritmo Genético

Entrada: Y, N
Saída: Solução

```
1 t=0;
2 inicia_população (Y, P, t);
3 para t = 1, ..., N faça
4 |   seleção_dos_pais (Y, P, t);
5 |   recombinação (P, t);
6 |   mutação (P, t);
7 |   avaliação (Y, P, t);
8 |   sobrevivem (P, t)
9 fim
```

em que Y é a função-objetivo, N é o número de gerações, t é o tempo atual e P a população.

2.3.8 Parâmetros Genéticos

Algoritmos Genéticos com parâmetros adaptativos (Algoritmos Genéticos Adaptativos, AGAs) são partes importantes e promissoras. É importante analisar de que maneira alguns parâmetros influem no comportamento dos Algoritmos Genéticos, para que se possa estabelecê-los conforme as necessidades do problema e dos recursos disponíveis:

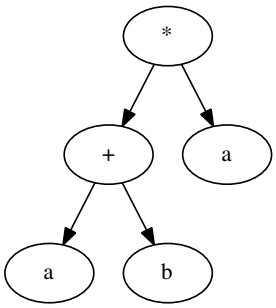
- **Tamanho da População**. O tamanho afeta o desempenho global e eficiência dos AGs. Com uma pequena população, tem-se uma pequena cobertura do espaço de busca do problema. Porém com uma grande população, é necessário maior poder computacional;
- **Taxa de Cruzamento**. Quanto maior essa taxa, mais rapidamente novas estruturas serão introduzidas. Porém, estruturas com aptidões boas serão removidas rapidamente. Com um valor baixo, o algoritmo pode tornar-se muito lento;
- **Taxa de Mutação**. Uma baixa taxa de mutação tenta evitar que uma população fique estagnado em um valor. Com alta taxa, a busca se torna aleatória;
- **Intervalo de Geração**. Controla a porcentagem da população que será substituída durante a próxima geração. Com um valor alto, pode ocorrer perda de estruturas de alta aptidão. Com um valor baixo, o algoritmo pode tornar-se muito lento.

2.4 Programação Genética

Uma outra abordagem bastante conhecida na área de otimização consiste na Programação Genética, do inglês *Genetic Programming* (GP) (KOZA, 1992), a qual é uma técnica meta-heurística evolucionista, dado que seu funcionamento é baseado na Teoria da Evolução de Darwin. Na prática, GP funciona de maneira bastante similar à uma outra técnica de otimização amplamente utilizada, Algoritmo Genético (KOZA, 1992). Ambas técnicas modelam problemas de otimização em estruturas que permitam, posteriormente, proporcionar novas soluções por meio de operadores de cruzamento (*crossover*), mutação e reprodução.

Enquanto que GA adota uma representação vetorial de cada solução, a qual é modelada por um agente do tipo cromossomo, no caso de GP cada agente (agora indivíduo de uma população) é codificado em uma estrutura de árvore binária, que possui dois tipos de nós: nós do tipo função e nós terminais. Imagine uma árvore de expressão, onde os operadores encontram-se representados pelos nós do tipo função, e os operandos são modelados pelos nós terminais (nós folha), conforme ilustra a Figura 12.

Figura 12 – Exemplo de árvore de expressão.



Fonte: Elaborada pelo autor.

Nesse caso, a solução $s \in \Re$ da expressão poderia ser obtida percorrendo-se a árvore, a qual codifica a seguinte expressão:

$$s = (a + b) * a, \tag{2.45}$$

onde $a, b \in \Re$. Suponha que um conjunto de pontos em $X = \{x_1, x_2, \ldots, x_m\}$, onde $x_i \in \Re$, para os quais deseja-se encontrar uma função $f : \Re \rightarrow \Re$ que melhor representa o comportamento desses pontos. Nesse caso, partindo de um conjunto de treinamento supervisionado $X^{Tr} \subset X$, isto é, um conjunto de dados onde $f(x_i)$, $\forall x_i \in X^{Tr}$ é conhecido, pode-se utilizar GP para encontrar a melhor árvore que representa essa função, onde "melhor" seria definido por alguma função objetivo, tal como o Erro Médio Quadrático, do inglês *Mean Squared Error*

(MSE):

$$MSE = \frac{1}{|X^{Tr}|} \sum_{i=1}^{|X^{Tr}|} (\hat{x}_i - x_i)^2, \tag{2.46}$$

onde $\hat{x}_i$ denota o valor estimado de x_i obtido por algum agente (árvore). Na prática, o agente que proporcionar o menor valor de MSE é então escolhido para estimar o restante dos valores do conjunto de teste $X \backslash X^{Tr}$.

A técnica de Programação Genética objetiva, então, a partir de um conjunto de árvores geradas aleatoriamente (**população**), utilizar os operadores citados anteriormente, ou seja, cruzamento, mutação e reprodução, visando produzir novos agentes para as próximas gerações (iterações do algoritmo). A ideia é baseada na premissa de que os agentes das novas populações serão mais "adaptados" ao problema, ou seja, eles proporcionarão melhores soluções. As próximas subseções apresentam os conceitos principais dos operadores genéticos.

2.4.1 Cruzamento

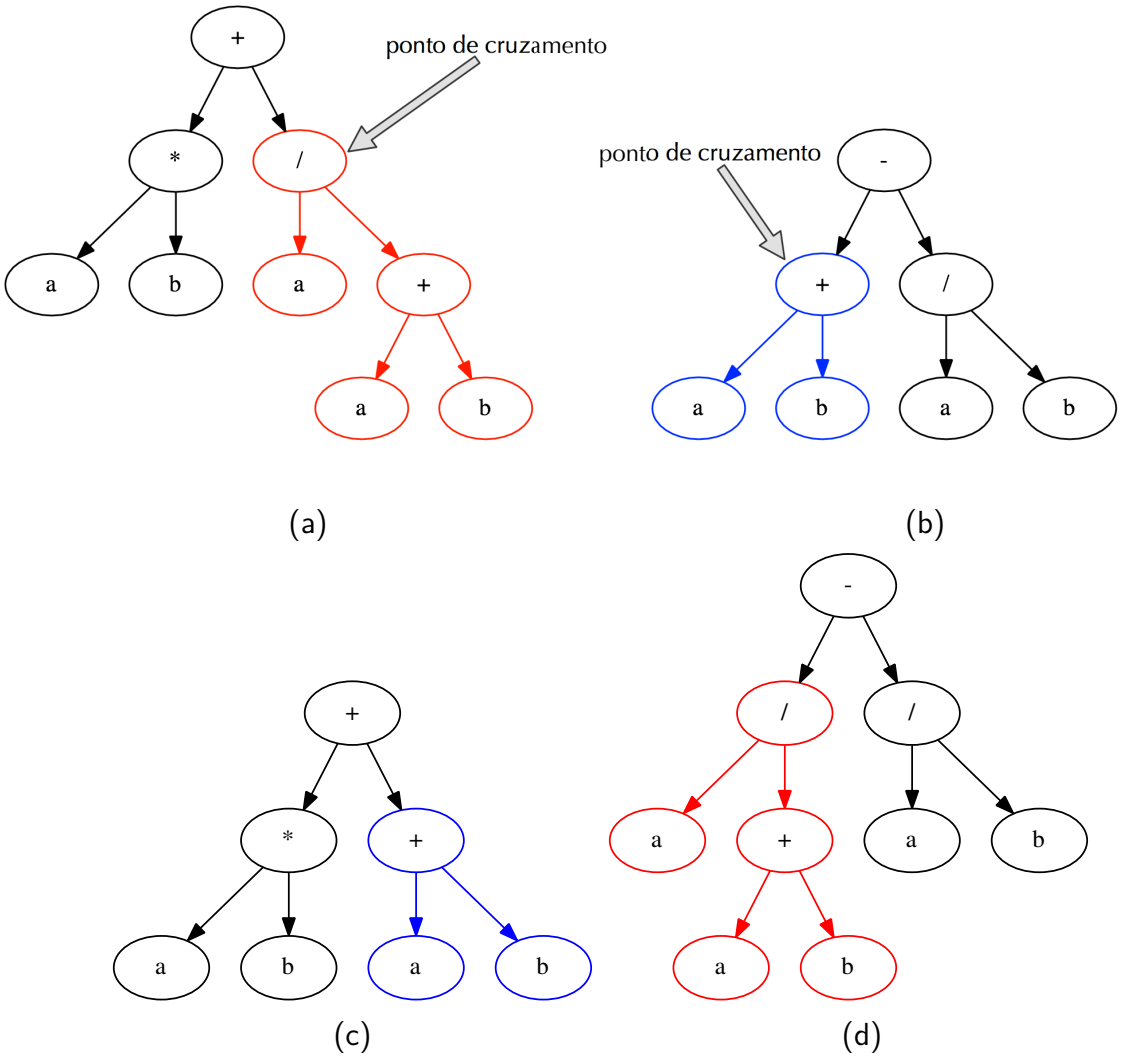
O operador de cruzamento funciona de maneira análoga à uma reprodução sexuada entre dois indivíduos, ou seja, existe uma troca de genes entre eles visando a geração de dois "filhos". Na prática, o cruzamento entre dois agentes consiste em trocar "pedaços" de suas respectivas árvores por meio do **ponto de cruzamento**, o qual indica o local de troca das informações. Em seguida, dois novos agentes são gerados por meio dessa operação para compor a nova população de agentes. A Figura 13 ilustra essa operação.

No caso ilustrado, a sub-árvore vermelha do agente 1 (Figura 13a) será posicionada no lugar da sub-árvore azul do agente 2 (Figura 13b) visando a geração do novo agente 2 (Figura 13d). Já a operação oposta será realizada com o intuito de gerar o novo agente 1 (Figura 13c). Note que essa operação utiliza um único ponto de cruzamento, pois também existem variantes que utilizam vários pontos de cruzamento (LINDEN, 2008).

2.4.2 Mutação

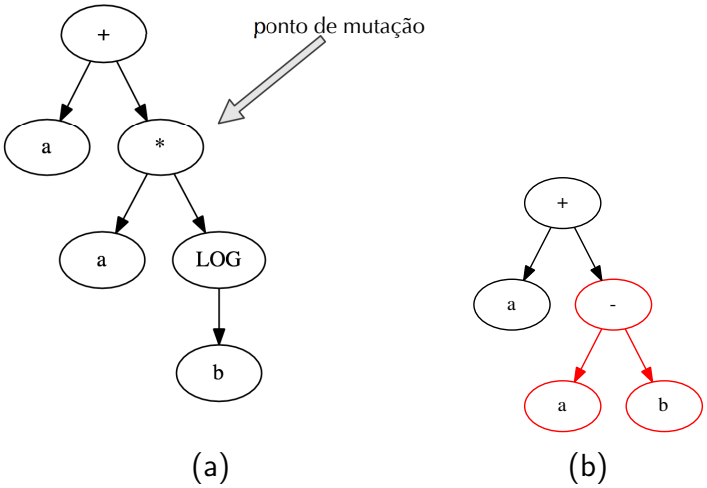
Já o operador de mutação tem por objetivo introduzir variabilidade genética nos agentes da população, porém sem efetuar o cruzamento com outros indivíduos. A ideia consiste na escolha aleatória de um **ponto de mutação** para, a partir dele, remover toda a sua sub-árvore e substituí-la por uma outra gerada aleatoriamente. A Figura 14 ilustra esse procedimento.

Figura 13 – Operação de cruzamento em uma árvore.



Exemplo de operação de cruzamento: (a) agente 1, (b) agente 2, (c) novo agente gerado 1 e (d) novo agente gerado 2. Fonte: Elaborada pelo autor.

Figura 14 – Operação de mutação em uma árvore.



Exemplo de operação de mutação: (a) agente 1 e o seu ponto de mutação, e (b) novo agente gerado. Fonte: Elaborada pelo autor.

De maneira análoga ao processo de cruzamento, também existe a opção da utilização de vários pontos de mutação (LINDEN, 2008). Já o operador de reprodução é bastante simples, e consiste apenas em copiar os elementos da geração atual para a próxima. Visando gerenciar todo esse processo, a técnica GP requer as taxas de probabilidade de ocorrência de cada uma das operações, bem como o usuário pode também escolher a probabilidade de ocorrência de cruzamento e mutação em nós do tipo função. Esse é um outro ponto interessante da técnica, pois um cruzamento a partir de um nó terminal geralmente não contribui muito para a variabilidade genética dos filhos, pois apenas esse nó será trocado com uma outra sub-árvore.

3 Metodologia

O projeto é descrito em 3 etapas principais: desenvolvimento da rede CRBM, aplicação de GP e GA nos hiperparâmetros da CRBM e, por fim, a análise descritiva dos resultados sobre os respectivos conjuntos de dados. A etapa inicial do projeto consistiu em estudar as propriedades e desenvolver a rede CRBM proposta por Lee et al. (2009). Os autores do algoritmo anterior relataram bons resultados em uma variedade de tarefas de reconhecimento visual (HUANG; LEE; LEARNED-MILLER, 2012) e dígitos (WICHT; HENNEBERTY, 2015). Em seguida, para desenvolver a Programação Genética e o Algoritmo Genético, foi utilizado o framework "*Distributed Evolutionary Algorithms in Python*" (FORTIN et al., 2012).

Assim, o problema de selecionar os hiperparâmetros é moldado considerando CRBMs na tarefa de reconstrução de imagem. Durante a etapa de aprendizagem de uma CRBM, optou-se por otimizar três hiperparâmetros: o tamanho do filtro, o número de filtros e os k-passos da amostragem de Gibbs. Inicialmente, apenas três foram selecionados pois são considerados os principais no contexto de CRBM e evita aumentar a complexidade do problema. O próximo passo diz respeito à inicialização do espaço de busca de técnicas meta-heurísticas com valores aleatórios dentro do intervalo de cada variável e, em seguida, executar o algoritmo GP e GA.

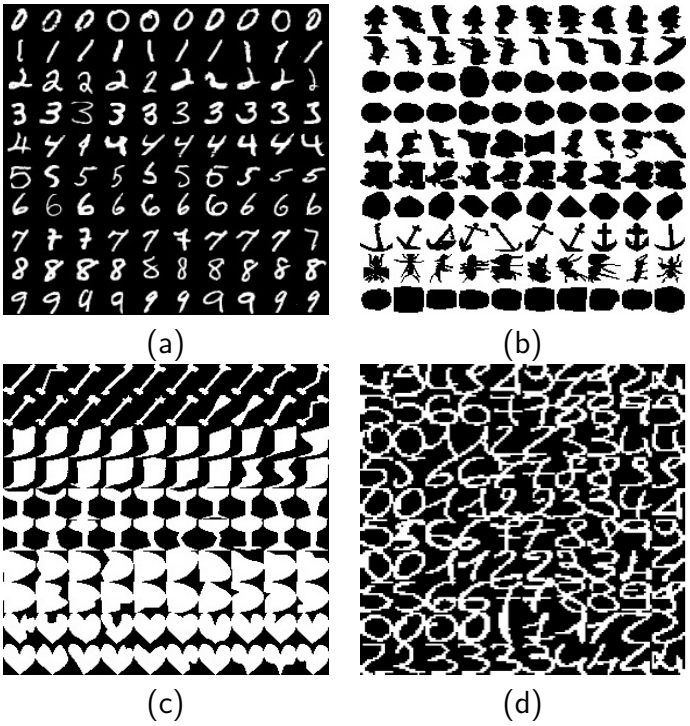
Para validar a proposta de reconstrução de imagens são propostas as seguintes base de dados:

- MNIST[1]: esta base de dados é composta por imagens de dígitos manuscritos. A sua versão original contém um conjunto de treinamento composto por 60.000 imagens dos dígitos '0'-'9', bem como um conjunto de teste composto por 10.000 imagens. Todas as imagens possuem a resolução de 28×28;
- CalTech 101 Silhouettes[2]: esta base de dados é baseada na famosa base Caltech 101, sendo composta por silhuetas de imagens distribuídas em 101 classes com resolução de 28×28;
- MPEG-7 Shape[3]: esta base de dados é composta por 1.400 imagens binárias distribuídas em 70 classes, sendo 20 imagens por classe. Como essas imagens possuem tamanhos variados, elas serão amostradas para um tamanho comum a todas, dado que uma RBM requer conjuntos de dados com o mesmo tamanho de entrada;
- Semeion[4]: Foram digitalizados 1593 dígitos manuscritos de cerca de 80 pessoas, esticados em uma caixa retangular 16×16 em uma escala de cinza de 256 valores. Em seguida,

[1] <http://yann.lecun.com/exdb/mnist/>
[2] <https://people.cs.umass.edu/~marlin/data.shtml>
[3] <http://www.dabi.temple.edu/~shape/MPEG7/dataset.html>
[4] <https://archive.ics.uci.edu/ml/datasets/semeion+handwritten+digit>

cada *pixel* de cada imagem foi dimensionado para um valor booleano (1/0) usando um limite fixo. Cada pessoa escreveu em um papel todos os dígitos de 0 a 9, duas vezes. O compromisso era escrever o dígito pela primeira vez da maneira normal (tentando escrever cada dígito com precisão) e pela segunda vez de maneira rápida (sem precisão).

Figura 15 – Exemplo de amostras de treinamento das bases.



(a) MNIST, (b) CalTech 101 Silhouettes, (c) MPEG-7 e (d) Semeion. Fonte: Elaborada pelo autor.

O critério para comparar o erro e a função custo foi utilizando a medida de Raiz do Erro Médio Quadrático (RMSE) entre a imagem de entrada e a imagem de saída de uma CRBM (crença da rede). Em geral, essa medida representa o desvio padrão das diferenças entre os valores estimados e os valores reais observados. Estas diferenças são chamadas de resíduos. A medida RMSE é comumente usada para expressar a acurácia dos resultados numéricos com a vantagem de que apresenta valores do erro nas mesmas dimensões da variável analisada e amplifica os grandes resíduos. O RMSE é definido por:

$$e_{RMS} = \sqrt{\frac{\sum_{i=1}^{n}(v_e^i - v_r^i)^2}{n}}, \tag{3.1}$$

onde v_r^i é o valor real da observação i e v_e^i é o valor estimado correspondente. Para este trabalho, foi utilizado um conjunto de validação extraído do conjunto de treinamento com o intuito de selecionar esse conjunto de parâmetros de maneira empírica, ou seja, similar a uma busca pseudo-ótima (*grid-search*).

Por fim, os resultados obtidos através dessas análises serão exibidos através de uma interface gráfica.

3.1 Configuração Experimental

Antes de iniciar os experimentos, é necessário definir os valores de probabilidade dos operadores, a quantidade de indivíduos e número de gerações. A Tabela 1 mostra a configuração de parâmetros de GA e GP definidas para este projeto.

Tabela 1 – Configuração dos parâmetros para este projeto.

Parâmetros	Valores
Taxa de cruzamento	0.4
Taxa de mutação	0.1
População por geração	30
Número de gerações	6

Fonte: Elaborada pelo autor.

Em Progamação Genética, é necessário definir também as funções dos operadores de inicialização, seleção, cruzamento e mutação. Para este projeto definiu-se a função *genHalfAndHalf* para inicialização, a função *selTournament* para seleção, a função *cxOnePoint* para cruzamento e a função *mutUniform* para mutação. Mais detalhes destas funções, encontram-se no Apêndice B.

Em Algoritmo Genético, é necessário definir a estrutura do cromossomo. Neste caso, é uma lista com 3 valores, onde cada índice representa um hiperparâmetro. Para a seleção definiu-se a função *selTournament*, similar a Programação genética, no qual três indivíduos da população são escolhidos aleatoriamente e deste grupo, o indivíduo que tiver menor função objetivo será escolhido. O método *cxTwoPoint* foi definido para o cruzamento, no qual dois trechos de dois indivíduos são selecionados e trocados pelo trecho equivalente ao outro. O resultado desta operação é um indivíduo que potencialmente combine as melhores características dos indivíduos usados como base. Finalmente, é necessário definir a função de mutação. Para cada índice, é gerado um valor aleatório entre 0 e 1. Caso esse valor seja menor que 10% e o gene ser um número inteiro, então é utilizado a função *numpy.random.randint* que gera um valor inteiro dado um intervalo. Se o gene for do tipo *float* então é utilizado a função *numpy.random.uniform* que retorna um valor real, usando uma distribuição uniforme.

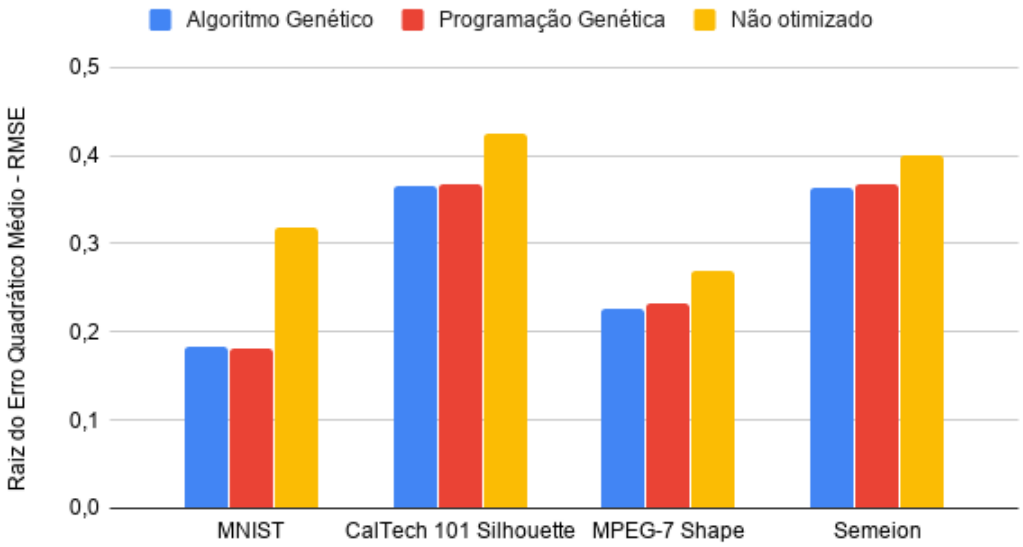
Para mais detalhes, verificar o Apêndice B que descreve como foi a implementação e apresenta mais informações das funções descritas nesta seção. O atual projeto encontra-se hospedado no github[5].

[5] <https://github.com/GuiCamargoX/OptCRBM>

4 Resultados

Este capítulo introduz os resultados experimentais da Máquina de Boltzmann Restrita Convolucional, bem como uma análise e discussão do comportamento das técnicas meta-heurísticas aplicadas ao seus hiperparâmetros. É possível notar que objetivo foi concluído; Algoritmo Genético e Programação Genética apresentaram resultados mais do que satisfatórios e um enorme potencial para otimizar mais do que três hiperparâmetros.

Figura 16 – Otimização de hiperparâmetros da CRBM.



O gráfico da Figura 16 nos mostra o menor erro, melhor indivíduo, que aparece durante a evolução de GA e GP do conjunto de validação: MNIST, CalTech 101, MPEG-7 e Semeion. A coluna azul representa o valor encontrado utilizando Algoritmo Genético, a coluna vermelha utilizando a Programação Genética e a coluna amarela representa o valor sem nenhum tipo de otimização. Os valores da classe "não otimizado" foram obtidos da média dos erros da geração 0, no qual o algoritmo esta avaliando árvores geradas aleatórias. É possível notar que em todos os conjuntos de dados, o Algoritmo Genético e Programação Genética apresentaram melhor desempenho comparado a nenhuma otimização.

Os resultados entre GA e GP, neste experimento, foram muitos semelhantes, porém GA obteve melhor resultado em três bases de dados. Pode-se observar que ele obteve os melhores nos conjuntos de dados CalTech, MPEG-7 e Semeion, enquanto o GP obteve o melhor resultado no conjunto de dados MNIST. As Tabelas 2, 3, 4, 5 , mostram os erros no conjunto de treinamento, conjunto de validação e no conjunto de teste após selecionado os melhores hiperparâmetros. Os valores encontrados no gráfico da Figura 16 são referentes ao erro no conjunto de validação enquanto os gráficos das Figuras 17, 18, 19 e 20 mostram a trajetória do erro ao longo das gerações. O eixo vertical representa a média do erro daquela geração e o

eixo horizontal apresenta o número da geração. A linha azul mostra a Programação Genética e a linha vermelha mostra o Algoritmo Genético. Notamos que ambos métodos convergem, são decrescentes e se mostram eficientes comparado a geração 0.

É relevante dizer que ao analisar a média do erro em cada geração, o Algoritmo Genético apresenta menor oscilação do que a Programação Genética. Isso deve-se a fato de que ao alterar um ramo da árvore em GP, resulta em uma modificação mais brusca nos valores dos hiperparâmetros. Em GA, como a estrutura é uma lista, o método utiliza os índices para alterar os valores dos hiperparâmetros, ou seja, só uma parte da estrutura é alterada, modificando levemente os valores. Porém, em outros projetos[1], ao modificar a estrutura de forma brusca, GP apresentou melhor desempenho pois consegue abranger mais resultados embora sua convergência possa demorar e também possa apresentar grandes oscilações em seus erros.

Tabela 2 – Valor dos erros no conjunto de dados MNIST.

	Algoritmo Genético	Programação Genética
Erro no conjunto de treinamento	0,1825	0,1833
Erro no conjunto de validação	0,1823	0,1818
Erro no conjunto de teste	0,1834	0,1840

Fonte: Elaborada pelo autor.

Tabela 3 – Valor dos erros no conjunto de dados Caltech.

	Algoritmo Genético	Programação Genética
Erro no conjunto de treinamento	0,3769	0,3776
Erro no conjunto de validação	0,3662	0,3667
Erro no conjunto de teste	0,3761	0,3719

Fonte: Elaborada pelo autor.

Tabela 4 – Valor dos erros no conjunto de dados MPEG.

	Algoritmo Genético	Programação Genética
Erro no conjunto de treinamento	0,2246	0,2296
Erro no conjunto de validação	0.2270	0.2318
Erro no conjunto de teste	0,2325	0,2351

Fonte: Elaborada pelo autor.

[1] FAPESP Proc. # 2018 / 10706-1

Tabela 5 – Valor dos erros no conjunto de dados Semeion.

	Algoritmo Genético	Programação Genética
Erro no conjunto de treinamento	0,3685	0,3700
Erro no conjunto de validação	0,3639	0,3666
Erro no conjunto de teste	0,3698	0,3736

Fonte: Elaborada pelo autor.

Figura 17 – Média do erro em cada geração utilizando o conjunto de dados MNIST

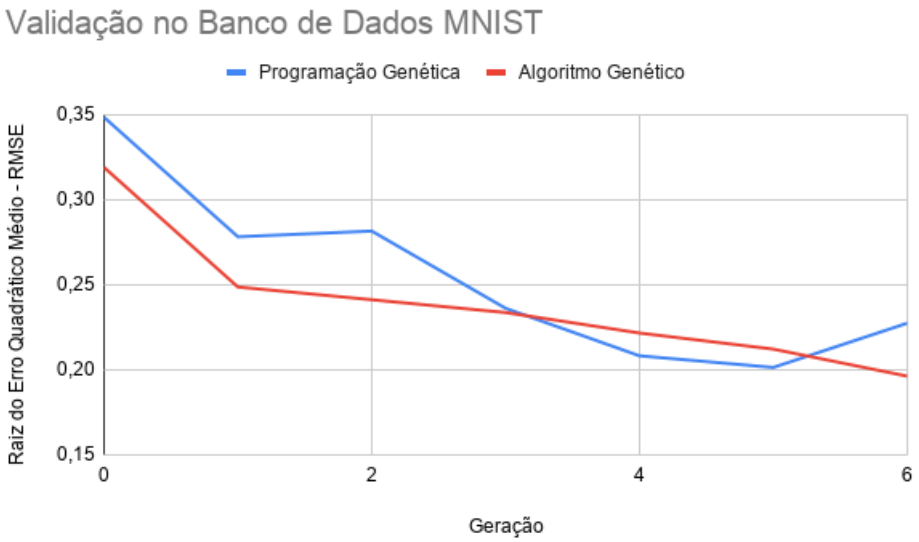


Gráfico mostra a média do erro da população em RMSE obtido em cada geração, validados no banco de dados MNIST.

Figura 18 – Média do erro em cada geração utilizando o conjunto de dados CalTech

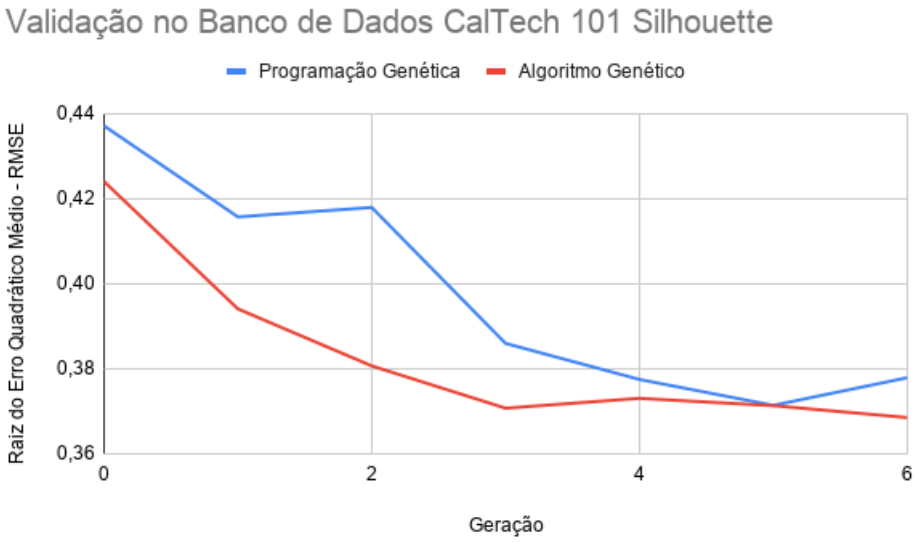


Gráfico mostra a média do erro da população em RMSE obtido em cada geração, validados no banco de dados CalTech.

Figura 19 – Média do erro em cada geração utilizando o conjunto de dados MPEG

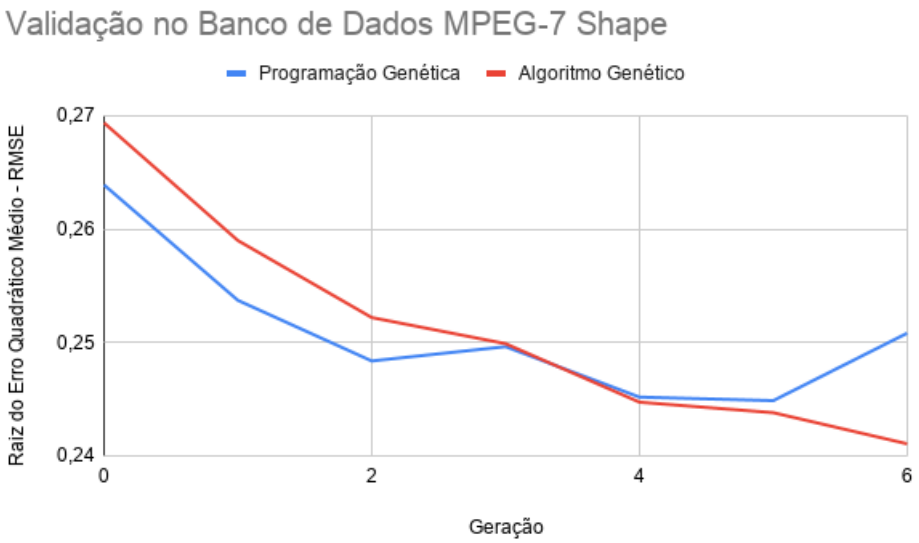


Gráfico mostra a média do erro da população em RMSE obtido em cada geração, validados no banco de dados MPEG.

Figura 20 – Média do erro em cada geração utilizando o conjunto de dados Semeion

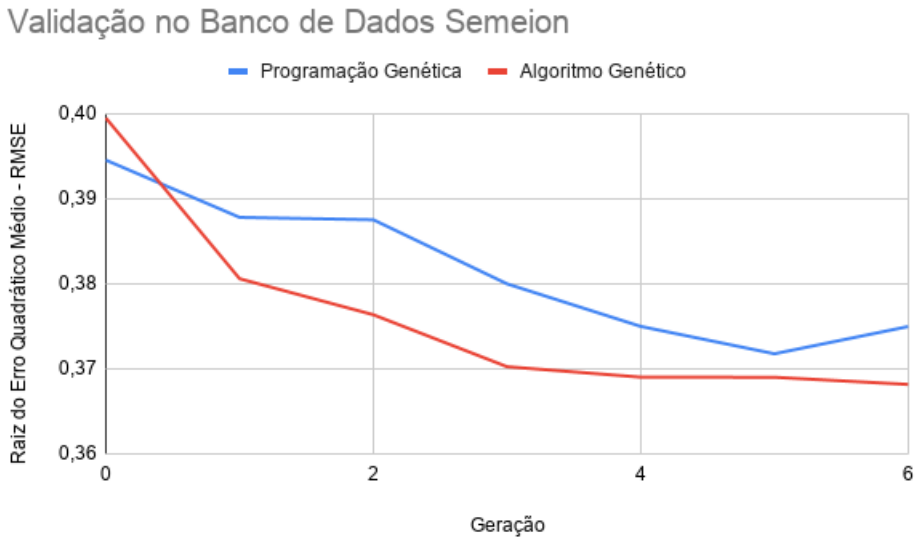


Gráfico mostra a média do erro da população em RMSE obtido em cada geração, validados no banco de dados Semeion.

5 Conclusão

Este trabalho apresentou uma aplicação da Programação Genética e Algoritmo Genético para selecionar hiperparâmetros viáveis no contexto de Máquinas de Boltzmann Restritas Convolucionais, que é um modelo probabilístico não supervisionado baseado em energia. A proposta obteve resultados adequados, porém requer um grande poder computacional, visto que avalia toda a população para cada geração. Ambos algoritmos evolutivos mostraram bom desempenho e resultados similares em otimizar os hiperparâmetros da CRBM. Algoritmo Genético se mostrou um pouco mais eficiente em achar o melhor indivíduo após as gerações enquanto a Programação Genética apresentou flutuações mais evidentes e piores resultados quando analisado a média dos erros de cada geração.

Devido a ser um problema simples, ou seja, três hiperparâmetros e limites dos intervalos reduzidos, optou-se por empregar apenas 6 gerações para a convergência da Programação Genética e Algoritmo Genético. Ao lidar com problemas mais complexos como otimizar todos os hiperparâmetros, um número maior de gerações pode ser necessário para gerar estruturas mais adequadas. Além disso, foi empregado o operador de cruzamento, que combina o código genético das árvores. Esta recombinação garante que os melhores indivíduos sejam capazes de trocar entre si as informações que os levam a ser mais aptos a manter melhores os valores, e assim gerar descendentes com tais características. Dos experimentos, o operador de cruzamento foi aplicado a uma taxa de 40% e observamos que uma operação com taxas acima dessa porcentagem, os valores flutuam bastante dificultando a convergência.

5.1 Trabalhos Futuros

Para trabalhos futuros, outras métricas podem ser analisadas como acurácia e pseudo-verossimilhança. Novos modelos de aprendizado de máquina podem ser introduzidos para validar a eficiência dos métodos GP e GA. Também é possível introduzir o conceito de validação cruzada com k-*fold* para calcular o desempenho da rede. Neste projeto, o conjunto de dados foi divido em três partes: conjunto de treinamento, conjunto de validação e conjunto de teste. O conjunto de dados de treinamento é utilizado para treinar a rede enquanto que o conjunto de validação é utilizado pelo algoritmos GP e GA para escolher os melhores hiperparâmetros que diminuem o erro e o conjunto de teste é utilizado para estimar o erro final. Finalmente, a introdução de uma nova função, Equação 5.1, que avalia o desempenho da rede pode ser proposta para resolver problema de *overfitting*, que ocorre quando um modelo estatístico se ajusta muito bem ao conjunto de dados anteriormente observado, mas se mostra ineficaz para prever novos resultados, e *underfitting* é o caso em que o modelo "não aprendeu o suficiente"

com os dados de treinamento, resultando em baixa generalização e previsões não confiáveis.

$$\min 100 \cdot (\mathrm{e}_{train} - \mathrm{e}_{valid})^2 + \mathrm{e}_{valid}. \tag{5.1}$$

A Equação 5.1 minimiza dois termos, o primeiro termo é dado pela diferença entre o erro no conjunto de treinamento e_{train} e o erro no conjunto de validação e_{valid} ao quadrado multiplicado por uma constante. O segundo termo é dado para minimizar o erro no conjunto de validação.

Referências

CHO, K.; ILIN, A.; RAIKO, T. Improved learning of gaussian-bernoulli restricted boltzmann machines. In: SPRINGER. *International conference on artificial neural networks*. [S.l.], 2011. p. 10–17.

FORTIN, F.-A.; De Rainville, F.-M.; GARDNER, M.-A.; PARIZEAU, M.; GAGNÉ, C. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, v. 13, p. 2171–2175, jul 2012.

HINTON, G. E. Training products of experts by minimizing contrastive divergence. *Neural computation*, MIT Press, v. 14, n. 8, p. 1771–1800, 2002.

HINTON, G. E. A practical guide to training restricted boltzmann machines. In: *Neural networks: Tricks of the trade*. [S.l.]: Springer, 2012. p. 599–619.

HINTON, G. E.; OSINDERO, S.; TEH, Y.-W. A fast learning algorithm for deep belief nets. *Neural computation*, MIT Press, v. 18, n. 7, p. 1527–1554, 2006.

HOLLAND, J. H. et al. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. [S.l.]: MIT press, 1992.

HUANG, G. B.; LEE, H.; LEARNED-MILLER, E. Learning hierarchical representations for face verification with convolutional deep belief networks. In: IEEE. *2012 IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.], 2012. p. 2518–2525.

KOREN, Y. The bellkor solution to the netflix grand prize. *Netflix prize documentation*, v. 81, n. 2009, p. 1–10, 2009.

KOZA, J. *Genetic programming: on the programming of computers by means of natural selection*. Cambridge, USA: The MIT Press, 1992.

KOZA, J. R.; ANDRE, D.; KEANE, M. A.; III, F. H. B. *Genetic programming III: Darwinian invention and problem solving*. [S.l.]: Morgan Kaufmann, 1999. v. 3.

LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFFNER, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, v. 86, n. 11, p. 2278–2324, 1998.

LEE, H.; GROSSE, R.; RANGANATH, R.; NG, A. Y. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In: ACM. *Proceedings of the 26th annual international conference on machine learning*. [S.l.], 2009. p. 609–616.

LINDEN, R. *Algoritmos Genéticos*. 2nd. ed. [S.l.]: Brasport, 2008.

MALLAWAARACHCHI, V. *Introduction to Genetic Algorithms — Including Example Code*. 2017. Disponível em: <https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3>. Acesso em: 07 maio 2020.

NIXON, M.; AGUADO, A. S. *Feature extraction and image processing for computer vision*. [S.l.]: Academic Press, 2012.

NOULAS, A. K.; KRÖSE, B. J. et al. Deep belief networks for dimensionality reduction. In: CITESEER. *Belgian-Netherlands Conference on Artificial Intelligence*. [S.l.], 2008. p. 185–191.

PLAHL, C.; SAINATH, T. N.; RAMABHADRAN, B.; NAHAMOO, D. Improved pre-training of deep belief networks using sparse encoding symmetric machines. In: IEEE. *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. [S.l.], 2012. p. 4165–4168.

PUMSIRIRAT, A.; YAN, L. Credit card fraud detection using deep learning based on auto-encoder and restricted boltzmann machine. *International Journal of advanced computer science and applications*, v. 9, n. 1, p. 18–25, 2018.

SALAKHUTDINOV, R.; MNIH, A.; HINTON, G. Restricted boltzmann machines for collaborative filtering. In: *Proceedings of the 24th international conference on Machine learning*. [S.l.: s.n.], 2007. p. 791–798.

SANTORO, A. Um pioneiro da física moderna e um dos criadores da mecânica estatística. *Revista Primus Vitam Nº*, v. 3, n. 2º, 2011.

SIEGEL, E. *Ask Ethan: How Can A Black Hole's Singularity Spin?* 2019. Disponível em: <https://medium.com/starts-with-a-bang/ask-ethan-how-can-a-black-holes-singularity-spin-36d7bf94e1ee>. Acesso em: 07 maio 2020.

THEANO. *Restricted Boltzmann Machines (RBM) Documentation*. 2019. Disponível em: <http://deeplearning.net/tutorial/rbm.html>. Acesso em: 07 maio 2020.

TULDER, G. van; BRUIJNE, M. de. Learning features for tissue classification with the classification restricted boltzmann machine. In: SPRINGER. *International MICCAI Workshop on Medical Computer Vision*. [S.l.], 2014. p. 47–58.

WICHT, B.; FISCHER, A.; HENNEBERT, J. Keyword spotting with convolutional deep belief networks and dynamic time warping. In: . [S.l.: s.n.], 2016. p. 113–120. ISBN 978-3-319-44780-3.

WICHT, B.; HENNEBERTY, J. Mixed handwritten and printed digit recognition in sudoku with convolutional deep belief network. In: IEEE. *2015 13th international conference on document analysis and recognition (ICDAR)*. [S.l.], 2015. p. 861–865.

Apêndices

APÊNDICE A – Derivadas dos parâmetros da RBM

A.1 Fase positiva

É possível calcular a derivada da função $\mathcal{F}$ em relação ao conjunto de parâmetros θ do modelo :

$$\frac{\partial \mathcal{F}(\mathbf{v})}{\partial \theta} = \frac{\partial}{\partial \theta}\left(-\ln\left(\sum_{\mathbf{h}\in\{0,1\}^n} \exp\left(-E(\mathbf{v},\,\mathbf{h})\right)\right)\right) \tag{A.1}$$

$$= -\left(\sum_{\mathbf{h}'\in\{0,1\}^n} \exp\left(-E(\mathbf{v},\,\mathbf{h}')\right)\right)^{-1}\left(\sum_{\mathbf{h}\in\{0,1\}^n} \frac{\partial(\exp\left(-E(\mathbf{v},\,\mathbf{h})\right))}{\partial\theta}\right) \tag{A.2}$$

$$= -\left(\sum_{\mathbf{h}'\in\{0,1\}^n} \exp\left(-E(\mathbf{v},\,\mathbf{h}')\right)\right)^{-1}\left(\sum_{\mathbf{h}\in\{0,1\}^n} -\exp\left(-E(\mathbf{v},\mathbf{h})\right)\cdot\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial\theta}\right) \tag{A.3}$$

Em seguida, dividimos o numerador e denominador por Z, de modo a aplicar as definições de distribuição conjunta e distribuição marginal:

$$= \left(\sum_{\mathbf{h}'\in\{0,1\}^n} \frac{1}{Z}\exp\left(-E(\mathbf{v},\,\mathbf{h}')\right)\right)^{-1}\left(\sum_{\mathbf{h}\in\{0,1\}^n} \frac{1}{Z}\exp\left(-E(\mathbf{v},\,\mathbf{h})\right)\cdot\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial\theta}\right) \tag{A.4}$$

$$= \left(\sum_{\mathbf{h}'\in\{0,1\}^n} P(\mathbf{v},\mathbf{h}')\right)^{-1}\left(\sum_{\mathbf{h}\in\{0,1\}^n} P(\mathbf{v},\mathbf{h})\cdot\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial\theta}\right) \tag{A.5}$$

$$= (P(\mathbf{v}))^{-1}\left(\sum_{\mathbf{h}\in\{0,1\}^n} P(\mathbf{v},\mathbf{h})\cdot\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial\theta}\right) \tag{A.6}$$

$$= \sum_{\mathbf{h}\in\{0,1\}^n} \frac{P(\mathbf{v},\mathbf{h})}{P(\mathbf{v})}\cdot\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial\theta} \tag{A.7}$$

$$= \sum_{\mathbf{h}\in\{0,1\}^n} P(\mathbf{h}|\mathbf{v})\cdot\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial\theta}. \tag{A.8}$$

Podemos escrever a equação acima como:

$$E_{h\sim P(\mathbf{h}|\mathbf{v})}\left[\frac{\partial E(\mathbf{v},\mathbf{H})}{\partial\theta}|\mathbf{v}\right] \tag{A.9}$$

em que **H** representa todo domínio $\{0,1\}^n$ possível. Nota-se que nesta fase, ocorre a amostragem das variáveis ocultas com os valores das variáveis visíveis fixadas.

A.2 Fase negativa

É possível mostrar o termo da fase negativa de outra maneira:

$$\sum_{\mathbf{v}\in\{0,1\}^m} P(\mathbf{v}) \cdot \frac{\partial \mathcal{F}(\mathbf{v})}{\partial \theta} = \sum_{\mathbf{v}\in\{0,1\}^m} P(\mathbf{v}) \sum_{\mathbf{h}\in\{0,1\}^n} P(\mathbf{h}|\mathbf{v}) \cdot \frac{\partial E(\mathbf{v},\mathbf{h})}{\partial \theta} \tag{A.10}$$

$$= \sum_{\mathbf{v}\in\{0,1\}^m} \sum_{\mathbf{h}\in\{0,1\}^n} P(\mathbf{v}) P(\mathbf{h}|\mathbf{v}) \cdot \frac{\partial E(\mathbf{v},\mathbf{h})}{\partial \theta} \tag{A.11}$$

$$= \sum_{\mathbf{v}\in\{0,1\}^m} \sum_{\mathbf{h}\in\{0,1\}^n} P(\mathbf{v},\mathbf{h}) \cdot \frac{\partial E(\mathbf{v},\mathbf{h})}{\partial \theta} \tag{A.12}$$

Podemos reescrever a equação da seguinte forma:

$$E_{v,h\sim P(\mathbf{v},\mathbf{h})}\left[\frac{\partial E(\mathbf{V},\mathbf{H})}{\partial \theta}\right]. \tag{A.13}$$

em que **V** e **H** representam todo domínio $\{0,1\}^m$e $\{0,1\}^n$, respectivamente.

A.3 Regra de Aprendizagem

Finalmente, é obtido :

$$-\frac{\partial \ln P(\mathbf{v}^{(t)})}{\partial \theta} = E_{h\sim P(\mathbf{h}|\mathbf{v})}\left[\frac{\partial E(\mathbf{v}^{(t)},\mathbf{H})}{\partial \theta}|\mathbf{v}^{(t)}\right] - E_{v,h\sim P(\mathbf{v},\mathbf{h})}\left[\frac{\partial E(\mathbf{V},\mathbf{H})}{\partial \theta}\right]. \tag{A.14}$$

Observe que o nome desses dois termos não veio de seu sinal, mas vem do fato de que a primeira fase $h \sim P(\mathbf{h}|\mathbf{v})$ é definida por θ e **h** no qual é calculado dado $\mathbf{v}^{(t)}$ pelo conjunto de treinamento. Já, a fase negativa $v, h \sim P(\mathbf{v},\mathbf{h})$, é definida por θ, sendo que **v** e **h** fazem parte da distribuição conjunta onde é necessário calcular todas as possiveis configurações.

A fase negativa é intratável. Imagine que v poderia ser um vetor binário de 100 dimensões. Então, em média, teríamos 2^{100} termos, o que é muito custoso. Assim, é preciso recorrer a métodos de amostragem, que são discutidos na seção tal.

APÊNDICE B – Implementação

Para desenvolver a Programação Genética e o Algoritmo Genético, iremos utilizar o framework "Distributed Evolutionary Algorithms in Python" (FORTIN et al., 2012). O DEAP é uma nova estrutura de computação evolutiva para prototipagem rápida e teste de ideias. Procura tornar os algoritmos explícitos e as estruturas de dados transparentes, funciona em perfeita harmonia com mecanismos de paralelização, como multiprocessamento e possui diversas características:

- Algoritmo Genético com diversas estruturas. Por exemplo, listas, árvores e vetores;
- Programação Genética com árvores de prefixo;
- Estratégias de evolução;
- Otimização multiobjetivo;
- Co-evolução (cooperativa e competitiva) de múltiplas populações;
- Paralelização das avaliações;
- Armazena os melhores indivíduos que viviam na população;
- Pontos de verificação que salvam instancias de um sistema regularmente;
- Módulo de *benchmarks* contendo as funções de teste;
- Genealogia de uma evolução;
- Exemplos de algoritmos alternativos: Otimização de enxame de partículas, Evolução diferencial, Algoritmo de estimativa de distribuição; e
- Fácil instalação com o comando: *pip install deap*.

O DEAP permite criar diversas estruturas de um algoritmo evolutivo, ou seja, em vez de limitar com tipos predefinidos, fornece maneiras de criar tipos adaptáveis para cada situação. Em vez de sugerir operadores, é pedido explicitamente que nós os escolhamos com sabedoria.

As próximas seções fornecem uma visão geral de como é simples implementar a otimização do problema de hiperparâmetros de uma CRBM com a Programação Genética usando DEAP.

B.1 Configurar parâmetros inciais

```
import sys
sys.path.append('../../CRBM/')
from crbm import CRBM

import operator
from sklearn.model_selection import train_test_split
from random import randint,uniform,seed
import numpy as np
import pandas as pd
import tensorflow as tf
import utils
import os

from deap import algorithms
from deap import base
from deap import creator
from deap import tools
from deap import gp

#Parâmetros fixos
model={

        # Tamanho do bloco para o max pooling
        'block_shape': (2, 2),

        # Parâmetro escasso -
        'pbias': 0.05,
        # tamanho do passo em direção a escarsidade
        'pbias_lambda': 5,

        # valor inicial  do viés escondido
        'init_bias': 0.01,
        # valor inicial  do viés visível
        'vbias': 0.001,
        # fator de regularização
        'regL2': 0.01,
        # Taxa de aprendizagem
```

```
        'epsilon': 0.1,
        # valor inicial e final do parâmetro
        #para controlar o efeito viés do vetor de entrada
         'sigma_start': 0.2,
         'sigma_stop': 0.1,

         # número de épocas
         'epoch_per_layer': 20,
         # tamanho do batch
         'batch':1000,
        }

# parameters
misc_params = {
    # caminho para salvar imagens de saída, gráficos de desempenho, etc.
    'results_dir': './results/',
    # nome do arquivo(pickle) para salvar a rede
    'pickle_fname': 'cdbn_net-natural_imgs.dump',
    # nome do arquivo para gerar erro do treinamento
    'err_fname_training': 'error_training.txt',
    # nome do arquivo para gerar erro do teste
    'err_fname_test': 'error_test.txt',
    }
```

O primeiro passo é importar todos os módulos necessários. A implementação da rede neural encontra-se no diretório /CRBM. Após, é necessário definir algumas estruturas como os hiperparâmetros do modelo que serão "fixos", ou seja, não serão otimizados e definir alguns nomes de arquivos de saída. É necessário definir tamanho do bloco, fatores de regularização, taxa de aprendizagem, número de épocas, entre outros.

B.2 Configurar intervalos e variáveis

```
#Intervalo (min,máx)
bound = {   "num_bases" : (1,25), #número de filtros
            "btmup_window_shape" : (5, 15), #tamanho do filtro
            "CD_steps" : (1,3)} #número de k-passos do CD

### Variáveis dos operadores ###
```

```
7  p_cx = 0.4  # Probabilidade de cruzamento
8  p_mut = 0.1 # Probabilidade de mutação
9  n_pop = 30   # População
10 n_gen = 6   # Número de gerações
11
12 #seed
13 rd_state= 42
14
15 #Define função intervalar
16 def applyBound(value):#retorna uma lista
17     params = np.absolute(value).tolist()
18     i=0
19
20     for key,val in bound.items():
21         if isinstance(val[0], int) and isinstance(val[1], int):
22             params[i] = int(params[i])
23
24         params[i] = val[0] if params[i] < val[0] else params[i]
25         params[i] = val[1] if params[i] > val[1] else params[i]
26         i+=1
27
28     return params
```

É necessário definir um intervalo em que os valores dos hiperparâmetros podem assumir para evitar que o método estime valores muito altos. Assim, teremos que o algoritmo convergira em espaço de curto tempo. Como é possível ver, a quantidade de filtros é um número inteiro e no intervalo [1, 25]. O tamanho do filtro é uma matriz quadrada de ordem 2 em que a coluna e a linha estão em intervalos [5, 15] e são números inteiros. O número de k-passos da amostragem de Gibbs está em [1, 3] e também é um inteiro.

Após definido os intervalos, é necessários atribuir valores aos operadores da programação genética. Neste caso, a probabilidade de mutação é 40%, a probabilidade de mutação é 10%, o número de indivíduos por geração é igual a 30 e o número de gerações é igual a 6. Para ser passível de reprodução é necessário configurar uma variável chamada *random state*, na linha 13, que executa o mesmo estado e podemos obter os mesmos resultados.

É necessário criar uma função, linha 16, que aplica a restrição no valor para mantê-lo no intervalo predefinido. Caso um número seja maior que valor máximo extremo do intervalo, então o número assumirá o valor extremo do intervalo. Podemos fazer isso analogamente caso o número seja menor que o mínimo do intervalo.

B.3 Carregar Banco de Dados - MNIST

```
#Banco de Dados
mnist = tf.keras.datasets.mnist
(X, Y),(X_test, Y_test) = mnist.load_data()

from sklearn.model_selection import train_test_split
X_train, X_valid, Y_train, Y_valid = train_test_split(X, Y,
                            stratify =Y, test_size=0.20, random_state=42)

X_train = X_train.astype(np.float)/255
X_valid = X_valid.astype(np.float)/255
X_test = X_test.astype(np.float)/255
```

O conjunto de dados MNIST é carregado diretamente do TensorFLow. As imagens são um *array NumPy* de 28x28. As *labels* (alvo da classificação) são um *array* de inteiros, no intervalo de 0 a 9. O comando *shape* retorna uma tupla (60000,28,28), ou seja, existem 60000 imagens no conjunto de treinamento, e cada imagem é representada em 28 x 28 pixels. Se inspecionarmos a primeira imagem do conjunto de treinamento, veremos que os valores dos pixels estão entre 0 e 255, então é necessário dividirmos os valores por 255 para escalarmos no intervalo de 0 e 1 antes de alimentar o modelo da rede neural. Para não termos um erro de generalização enviesado, iremos reservar uma parte do conjunto de treinamento para validar os nossos modelos e apenas no final, com os hiperparâmetros escolhidos é estimado o erro utilizando o conjunto de teste.

B.4 Configurar as primitivas da árvore

```
pset = gp.PrimitiveSet("main", 0)
pset.addPrimitive(np.add, arity=2)
pset.addPrimitive(np.subtract, arity=2)
pset.addPrimitive(np.negative, arity=1)
pset.addEphemeralConstant("rand101",
  lambda: [randint(bound["num_bases"][0],bound["num_bases"][1]),
  randint(bound["btmup_window_shape"][0],bound["btmup_window_shape"][1]),
  randint(bound["CD_steps"][0],bound["CD_steps"][1]) ] )
```

Primeiramente, as primitivas das árvores são os nós internos. Então é necessário definir quais operadores este pode assumir. Para este projeto, iremos utilizar apenas os operadores de

soma, subtração e negação. Outros operadores como logaritmo, seno, divisão, multiplicação, mostraram que tendem a gerar valores extremos do intervalo predefinido.

A estrutura da árvore é fornecida no módulo gp, pois é um dos poucos tipos de dados que a biblioteca padrão do Python não fornece. A listagem a acima apresenta uma instanciação do conjunto primitivo com os operadores básicos fornecidos pelo pacote *Numpy*.

A primeira linha cria um conjunto primitivo. Seus argumentos são o nome do procedimento que ele irá gerar ("main") e seu número de entradas, 0, visto que a função não retorna nenhum valor. As próximas três linhas adicionam funções como primitivas. O primeiro argumento é a função a ser adicionada e o segundo argumento a função aridade.

Por último, é acrescentado uma constante efêmera(*ephemeral constant*), que é um nó terminal mas é gerado a partir de uma determinada função no tempo de execução. Neste caso, a função lambda retornará uma lista com os valores do hiperparâmetros, dentro do intervalo predefinido. A função *randint*, retorna um número inteiro aleatório com a distribuição "uniforme discreto" do especificado intervalo, e a função *uniform* retorna um número real com densidade uniforme dentro de um intervalo. O valor da constante efêmera é determinado quando é inserido na árvore e nunca muda, a menos que seja substituído por outra constante efêmera.

B.5 Definir indivíduo e população

```
1  #cria classe indivíduo
2  creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
3  creator.create("Individual", gp.PrimitiveTree,
4                                          fitness=creator.FitnessMin)
5
6  toolbox = base.Toolbox()
7
8  # Registra atributo
9  toolbox.register("expr", gp.genHalfAndHalf, pset=pset, min_=1,
10                                                          max_=2)
11
12 # Registra indivíduo e população
13 toolbox.register("individual", tools.initIterate, creator.Individual,
14                    toolbox.expr)
15 toolbox.register("population", tools.initRepeat, list,
16                    toolbox.individual)
17 toolbox.register("compile", gp.compile, pset=pset)
```

A primeira coisa a fazer é pensar no tipo apropriado para o seu problema. Então, em vez de procurar na lista de tipos disponíveis, o DEAP permite que você crie seu próprio. Isso é feito com o módulo *creator*, que permite criar classes que atendam às necessidades dos algoritmos evolutivos. Criar um tipo apropriado pode parecer difícil, mas o *creator* facilita muito. De fato, isso geralmente é feito em uma única linha. Utilizamos o *creator* na linha 2 e linha 3. Na linha 2, é criado a classe 'FitnessMin' que descende da classe *base.Fitness* e atribuímos a variável *weights* uma tupla com valor (-1.0,), indicando que o problema é de minimização e possui apenas uma função objeto. Na linha 3, é criado a classe 'Individual' que herda as propriedades de uma árvore primitiva e possui um atributo *fitness* do tipo *FitnessMin* que acabamos de criar. Isso é usado por alguns operadores de gp para modificar os indivíduos.

O código apresentado na seção produz árvores válidas. No entanto, essas árvores ainda não são indivíduos válidos para a evolução. É preciso combinar o *creator* e a *tools* para produzir indivíduos válidos. O módulo *tools* pretende conter todas as ferramentas evolutivas, dos inicializadores de objeto ao operador de desempenho e permite fácil configuração de cada algoritmo. O módulo possui basicamente dois métodos, register() e unregister(), usados para adicionar ou remover qualquer função. As funções que inicializam indivíduos e populações são registradas no módulo *tools*. É possível chamar a função *toolbox.population()* para criar instantaneamente uma população ou chamar *toolbox.individual()* para retornar um individuo do tipo *PrimitiveTree*.

Para inicializar a árvore, registramos a função *gp.genHalfAndHalf* que gera uma expressão, linha 9, e passamos como parâmetro o conjunto de primitivas *pset*, além dos valores de mínimo e máximo de altura que a arvore pode atingir. Essa função gera uma árvore do tipo *PrimitiveTree* em que na metade do tempo, a expressão é gerada com a função *genGrow()*, na outra metade, a expressão é gerada com *genFull()*.

A função registrada *toolbox.individual*, quando chamada, usará a função *tools.initIterate* disponibilizada no módulo de ferramentas para gerar uma instancia da classe *Individual* com o que é produzido pela função *toolbox.expr* definida anteriormente. O mesmo é feito para a função da população.

No DEAP, as árvores podem ser traduzidas para código Python legível e compiladas para objetos usando funções fornecidas pelo módulo gp. Para torná-lo executável, a função *gp.compile* transforma a árvore primitiva em sua forma executável, uma função Python, usando um conjunto primitivo *pset* fornecido como terceiro argumento da função de avaliação.

B.6 Definir função a minimizar

```
1 def evalAcc(individual):
2
```

```
    result = toolbox.compile(expr=individual)
    bound_result = applyBound(result)

    model['num_bases']= bound_result[0]
    model['btmup_window_shape']= ( bound_result[1], bound_result[1])
    model['CD_steps']=bound_result[2]

    # ---Processamento das imagens
    torch_batch = utils.process_array_to_pytorch(X_train,
        model['btmup_window_shape'], model['block_shape'] )
    torch_batch_valid = utils.process_array_to_pytorch(X_valid,
        model['btmup_window_shape'], model['block_shape'] )

    network = CRBM(model, (torch_batch.shape[2],torch_batch.shape[3],
        torch_batch.shape[1] ) )

    for epoch_idx in range(model['epoch_per_layer']):
        print("Training trial #%s.." % epoch_idx)


        for i in range(len(torch_batch)//network.batch):
            print("\n------ Epoch", epoch_idx, ", batch" , i,"------")
            b= torch_batch[i*model['batch']:i*model['batch']
                + model['batch'],:,:,:]
            print(i,i*model['batch'],i*model['batch']+model['batch'])
            network.contrastive_divergence(b)


        # Zerar os erros cometidos no treinamento
        network.epoch_err = []

        print("\n\n\nERROR Test")
        for i in range(len(torch_batch_valid)//network.batch):
            print("\n------ Epoch", epoch_idx, ", batch" , i,"------")
            b= torch_batch_valid[i*model['batch']:i*model['batch']
                +   model['batch'],:,:,:]
            print(i,i*model['batch'],i*model['batch']+model['batch'])
            network.gibbs(b)

```

```
42          # calcular a média do erro do teste cometido
43          # e salvá-lo no arquivo
44          mean_err = np.mean(network.epoch_err)
45          # Zerar os erros cometidos no teste
46          network.epoch_err = []
47
48
49          # parar o decaimento depois de algum ponto
50          if network.std_gaussian > network.model['sigma_stop']:
51               network.std_gaussian *= 0.99
52
53      return mean_err,
```

Antes de prosseguirmos, precisamos definir qual função queremos minimizar. A função acima evalAcc é responsável por calcular erro quadrático médio da raiz do conjunto de validação com os parâmetros obtidos da árvore. Conforme visto na seção anterior, para tornar a árvore em executável, precisamos compilar a expressão com a função *toolbox.compile()*, linha 3. Essa função contém apenas um argumento: a árvore/expressão a compilar.

Após compilado o código em um objeto que pode ser chamado, ele retorna a lista com o resultado da expressão obtida pela árvore. Antes de utilizarmos esse resultado como hiperparâmetros da CRBM, é necessário chamar a função *applyBound()*, linha 5, criada anteriormente, para estabelecer os intervalos e evitar valores negativos.

Finalmente, iniciamos o processamento das imagens e instanciamos o objeto CRBM com os novos valores, ajustando os valores com o conjunto de dados de validação. O processamento transforma as imagens do tipo *array* para *tensor*, estrutura de dados essencial do *pytorch*.

B.7 Definir operadores evolutivos

```
1  toolbox.register("evaluate", evalAcc )
2  toolbox.register("select", tools.selTournament, tournsize=3)
3  toolbox.register("mate", gp.cxOnePoint)
4  toolbox.register("expr_mut", gp.genFull, min_=0, max_=2)
5  toolbox.register("mutate", gp.mutUniform, expr=toolbox.expr_mut,
6                              pset=pset)
7
8  toolbox.decorate("mate",
9                   gp.staticLimit(key=operator.attrgetter("height"),
```

```
10                  max_value=17))
11 toolbox.decorate("mutate",
12                  gp.staticLimit(key=operator.attrgetter("height"),
13                  max_value=17))
```

Os operadores são como inicializadores, exceto que alguns já estão implementados no módulo *tools*. Depois de escolher os operadores desejados, basta registrá-los utilizando a função *toolbox.register()*. O primeiro parâmetro de entrada é uma string que representa o operador. O segundo parâmetro é a referência a uma função em que será chamada quando o *toolbox* executar o operador. Os demais parâmetros são atribuições a valores que deseja da função. Os nomes usuais para as ferramentas evolutivas são mate(), mutate(), evaluate() e select(), no entanto, qualquer nome pode ser registrado, desde que seja único.

Na linha 1, estamos registrando a função objetivo, informando a função *evalAcc* como parâmetro. Na linha 2, é registrado o operador de seleção. Neste caso, escolhemos a função Seleção por Torneio que já esta implementada dentro do módulo *tools*. Atribuímos o tamanho do torneio igual a 3, ou seja, dentre 3 indivíduos aleatórios selecionados da população, é selecionado o que possui menor erro para prosseguir. Na linha 3, é registrado o operador de cruzamento *cxOnePoint*, implementado pelo módulo gp, onde seleciona aleatoriamente um ponto que define uma subárvore de cada indivíduo e realiza a troca entre elas. Na linha 5, registramos o operador de mutação, no qual é selecionado aleatoriamente um ponto no indivíduo da árvore e substituído a subárvore nesse ponto pela expressão gerada usando o método *expr()*, registrado na linha 4. A função *gp.genFull* gera uma expressão em que cada folha tenha a mesma profundidade entre o mínimo e máximo fornecidos como parâmetros, neste caso, 0 e 2 respectivamente.

Um recurso poderoso que ajuda a controlar a precisão durante a evolução sem alteração no algoritmo ou nos operadores é a função *toolbox.decorate()*. Na linha 8 e 11, limitamos a altura dos indivíduos gerados por cruzamento e mutação. Isso é realizado para evitar uma importante desvantagem da Programação Genética: a expansão. Koza em seu livro sobre Programação Genética (KOZA et al., 1999) sugere usar uma profundidade máxima de 17.

B.8 Executar a Programação Genética

```
1 def run_GP():
2
3     random.seed(rd_state)
4
5     pop = toolbox.population(n=n_pop)
```

```
6     hof = tools.HallOfFame(1)
7
8     stats = tools.Statistics(lambda ind: ind.fitness.values)
9     stats.register("avg", np.mean)
10    stats.register("std", np.std)
11    stats.register("min", np.min)
12    stats.register("max", np.max)
13
14    pop, log = algorithms.eaSimple(pop, toolbox, cxpb=p_cx,
15                                   mutpb=p_mut, ngen=n_gen,
16                                   stats=stats, halloffame=hof,
17                                   verbose=True)
18
19    hof[0].param= applyBound( toolbox.compile(hof[0]) )
20    return pop, log, hof[0]
```

O DEAP vem com várias ferramentas de suporte que podem ser facilmente integradas a qualquer implementação. Esta seção apresenta alguns deles no contexto de otimização de hiperparâmetros.

A primeira ferramenta, Estatística(*Statistics*), calcula estatísticas sobre os atributos arbitrários de objetos designados, geralmente os indivíduos. O atributo é especificado por uma função-chave na instanciação do objeto de estatísticas, antes de iniciar o algoritmo. Essa chave deve receber uma função que será aplicada posteriormente aos dados. O exemplo de código acima, linha 8, usa o atributo *fitness.values* de cada elemento.

As linhas entre 9 e 12 registram os operadores de estatísticas, esse método recebe uma identificação como primeiro argumento e uma referência a uma função que opera em vetores como segundo argumento. Qualquer argumento subsequente é passado para a função quando chamado. A criação do objeto *Statistics* está concluída.

Na linha 5 e 6, configuramos o tamanho da população executando a função *toolbox.population(n)* e configuramos como 1 o número de indivíduos para manter no *HallOfFame*. O *HallOfFame* preserva os melhores indivíduos que apareceram durante uma evolução. A cada geração, ele varre a população e salva os indivíduos em um arquivo separado que não interage com a população. Caso a melhor solução desapareça durante a evolução, ela ainda estará disponível no hall da fama. A classe *HallOfFame* fornece uma interface semelhante a uma lista. É possível recuperar seu comprimento, iterá-lo para frente, para trás e obter um item ou uma fatia dele.

Ao usar um algoritmo predefinido como *eaSimple()*, linha 14, o objeto estatístico criado

anteriormente pode ser fornecido como argumento. As estatísticas serão automaticamente avaliadas na população a cada geração. O argumento *verbose* imprime as estatísticas na tela enquanto a otimização ocorre. Quando o algoritmo termina, a população final e um *Logbook* são retornados. O *Logbook* é uma sequência cronológica de registros de cada geração, como dicionários.

O hall da fama propõe uma interface de lista em que os indivíduos são classificados em ordem decrescente em relação ao seu desempenho. Assim, a solução mais adequada pode ser recuperada acessando o primeiro elemento da lista. Na linha 19, é possível ver que compilamos o primeiro da lista e em seguida aplicamos a função *applyBound()* para descobrir o intervalo e salvar em um atributo chamado 'param' pertencente a hof[0].

Finalmente, essa função retorna os indivíduos da população referentes a última geração, o *LogBook* e o melhor indivíduo encontrado. É possível imprimir o *Logbook* na tela ou arquivo e assim, gerar gráficos de comparação de dados entre cada geração. A tabela 6 ilustra os resultados obtidos pelo *LogBook*.

Tabela 6 – Resultados obtidos do *LogBook*

gen	avg	min	max	std
0	108,2308	74,4077	196,3401	36,1028
1	89,3713	74,4077	132,6445	18,7059
2	78,2968	72,7590	96,8404	5,9608
3	78,0964	72,7590	122,8168	10,8521
4	91,3146	72,4071	298,8567	56,3664
5	78,8239	72,4071	135,4203	14,2297
6	81,8763	72,4071	197,2321	23,3652
7	76,3764	72,4071	152,1609	14,1184
8	73,3509	72,0611	76,3543	1,0659
9	78,8183	72,0611	141,2546	18,0711
10	76,5342	72,0611	108,8666	9,2779

Estatística descritiva; A primeira coluna fornece o número da geração e a segunda mostra a média do erro. A terceira e quarta coluna fornecem o erro mínimo e máximo de cada geração, respectivamente. A quinta coluna mostra o desvio padrão. Fonte: Elaborado pelo Autor.

APÊNDICE C – Interface Gráfica

A proposta ao criar uma interface gráfica é permitir que o usuário manipule os valores dos parâmetros e assim, estudar diversas configurações possíveis e entender como o processo é realizado. Este Apêndice possui duas seções: a seção C.1 nos apresenta uma pequena introdução ao PyQt, ferramenta utilizada para desenvolver a interface gráfica; a seção C.2 nos apresenta as funcionalidades e imagens do aplicativo final.

C.1 PyQt

Qt é um conjunto de bibliotecas C++ de plataforma cruzadas que implementam APIs de alto nível. Com o Qt, é possível acessar muitos aspectos de sistemas móveis e de desktop modernos. Isso inclui serviços de localização e posicionamento, conectividade multimídia, NFC e Bluetooth, um navegador da web baseado no Chromium e o desenvolvimento tradicional de interface gráficas.

O PyQt utiliza-se de uma das características principais do Python. Em uma forma mais simplificada, o PyQt nada mais é do que a junção do útil ao agradável. No qual detém todo o poder e riqueza do Qt, utilizando toda a praticidade e produtividade que o Python oferece ao desenvolvedor. Ele é implementado com mais de 35 módulos de extensão e permite que a linguagem de programação Python seja usado como uma alternativa ao C++ em desenvolvimento de aplicativos para outras plataformas suportadas, incluindo iOS e Android. Podemos destacar aplicações, como a Eric IDE e a Spyder IDE, relizadas utilizando PyQt. Outra aplicação de muito destaque no Python e que utiliza o PyQt é a Ninja IDE. Esses são poucos exemplos de uma série de outras aplicações que o utilizam.

O PyQt5 também pode ser incorporado em aplicativos baseados em C++ para torna los mais eficientes, permitindo que os usuários configurem ou aprimorem a funcionalidade dos aplicativos.

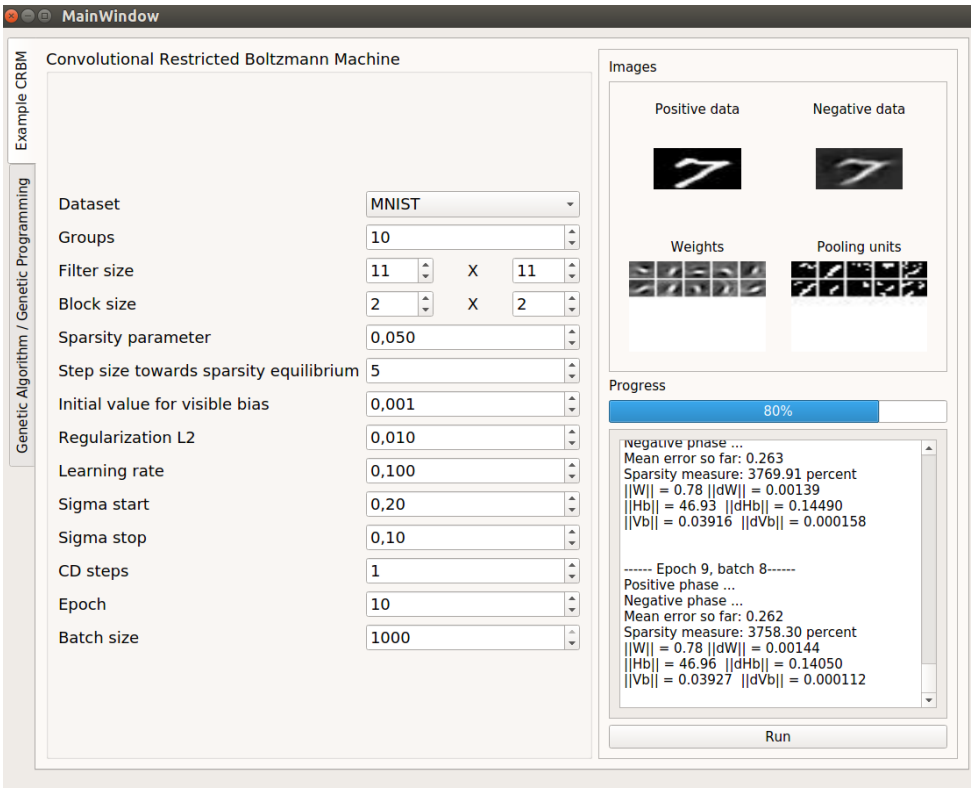
C.2 Aplicativo

O aplicativo consiste em dois painéis principais: um painel, nomeado *Example CRBM*, para manipular apenas os hiperparâmetros da Máquina de Bolztmann Restrita Convolucional e o outro, nomeado *Genetic Algorithm/Genetic Programming*, para manipular os parâmetros dos métodos Programação Genética e Algoritmo Genético.

O primeiro painel foi desenvolvido para testar diversas configurações de hiperparâmetros da Máquina de Boltzmann Restrita Convolucional, selecionando-os manualmente. Ele pode ser

visualizado na Figura 21 e é constituído por três *cards*: configuração, imagens, e uma área de texto. Os hiperparâmetros estão localizados no *card* configuração, ao lado esquerdo da figura, onde é possível selecionar o conjunto de dados e hiperparâmetros como a quantidade de grupos e o tamanho do filtro de convolução. O *card* imagens contém as imagens resultantes da rede neural como imagens dos pesos e da camada de pooling. Além disso, nesse mesmo *card*, é gerado a imagem do dado negativa que representa a reconstrução da imagem utilizada nos parâmetros de entrada. Finalmente, o *card* texto nos mostra o log dos resultados. Com ele é possível rastrear o progresso do treinamento.

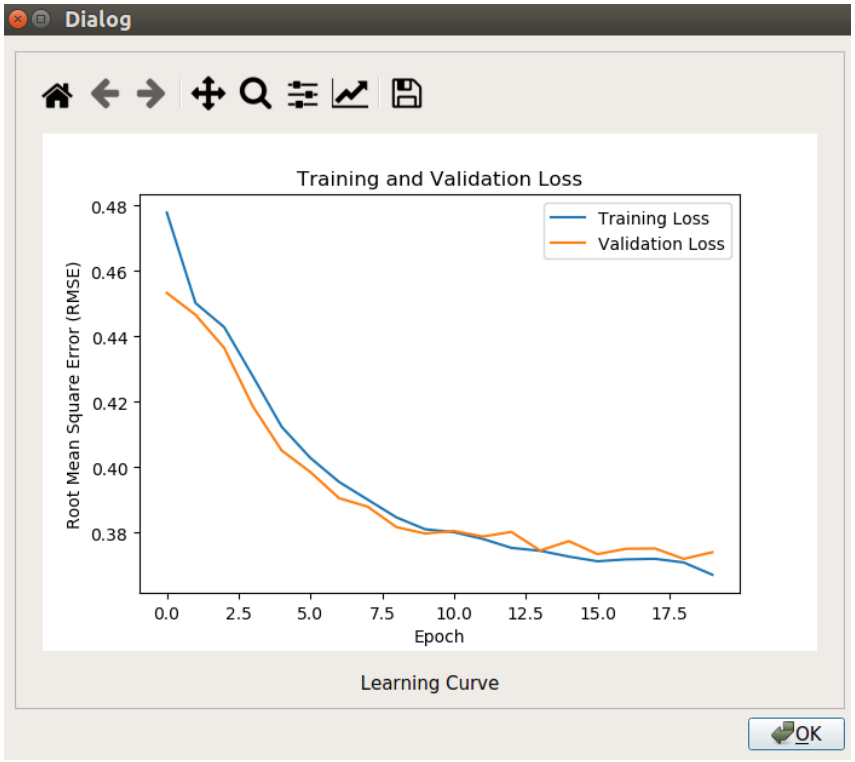
Figura 21 – Painel 1



Fonte: Elaborado pelo autor

Após finalizado o treinamento da rede, a barra progresso mostrará 100% e uma janela com um gráfico mostrando as curvas de aprendizagem ira aparecer. Conforme é possível ver na Figura 22, a janela contém opções para o usuário interagir como salvar, ampliar e mover a imagem gerada. As curvas de aprendizagem mostram o erro médio quadrático por época, o que nos permite acompanhar a evolução e notar casos de *underfitting* e *overfitting*.

Figura 22 – Gráfico mostrando as curvas de aprendizagem

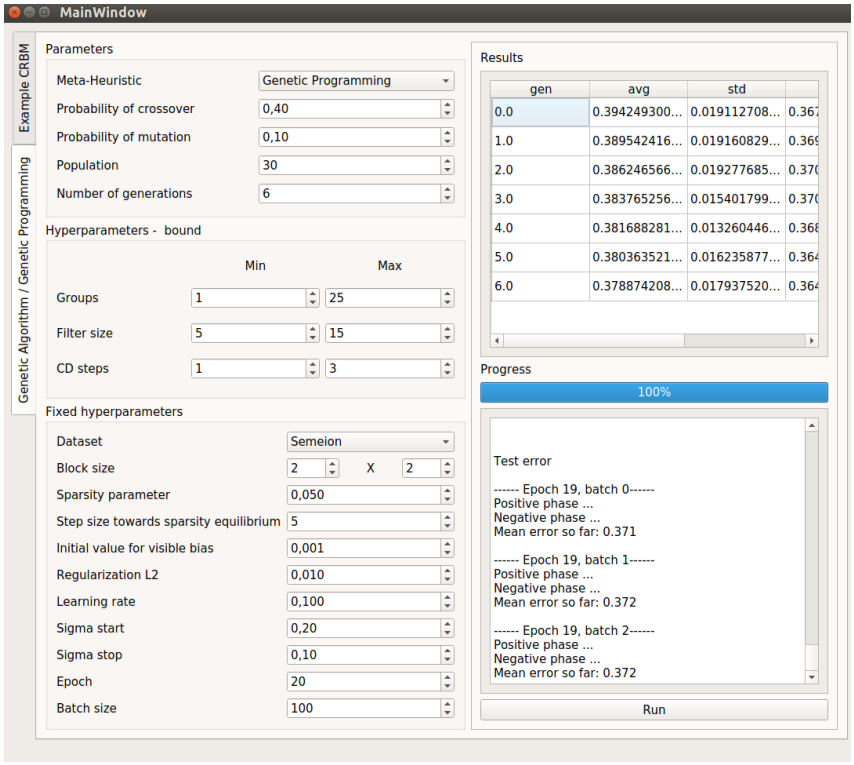


Fonte: Elaborado pelo autor

O segundo painel é apresentado pela Figura 23. Com ele é possível selecionar o método meta-heurístico e seus parâmetros como a taxa de probabilidade de mutação; definir o intervalo do domínio dos hiperparâmetros que serão otimizados; definir os hiperparâmetros fixos, aqueles que não irão ser otimizados; verificar a tabela de resultados que nos mostra a média, desvio padrão, mínimo e máximo de cada geração; e a área texto que nos mostra o progresso do treinamento.

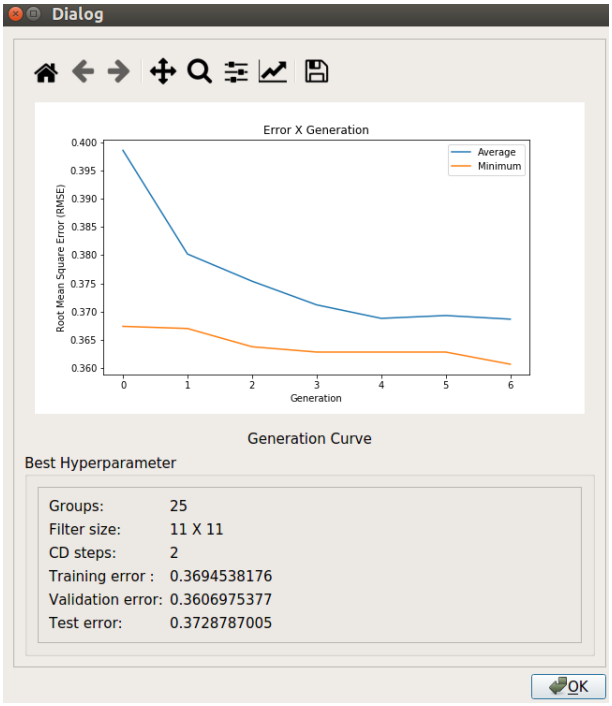
A Figura 24 nos mostra a janela gerada após o método meta-heurístico terminar. Ela apresenta as mesmas opções que a Figura 22, porém o gráfico nos mostra a média e o menor valor encontrado em cada geração. Além disso, é apresentado a melhor configuração de hiperparâmetros e também seu erro de treinamento, erro de validação e erro de teste.

Figura 23 – Painel 2



Fonte: Elaborado pelo autor

Figura 24 – Curva de geração



Fonte: Elaborado pelo autor